

AllFusion[™] Harvest Change Manager

User Guide

5.11



Computer Associates[™]

This documentation and related computer software program (hereinafter referred to as the "Documentation") is for the end user's informational purposes only and is subject to change or withdrawal by Computer Associates International, Inc. ("CA") at any time.

This documentation may not be copied, transferred, reproduced, disclosed or duplicated, in whole or in part, without the prior written consent of CA. This documentation is proprietary information of CA and protected by the copyright laws of the United States and international treaties.

Notwithstanding the foregoing, licensed users may print a reasonable number of copies of this documentation for their own internal use, provided that all CA copyright notices and legends are affixed to each reproduced copy. Only authorized employees, consultants, or agents of the user who are bound by the confidentiality provisions of the license for the software are permitted to have access to such copies.

This right to print copies is limited to the period during which the license for the product remains in full force and effect. Should the license terminate for any reason, it shall be the user's responsibility to return to CA the reproduced copies or to certify to CA that same have been destroyed.

To the extent permitted by applicable law, CA provides this documentation "as is" without warranty of any kind, including without limitation, any implied warranties of merchantability, fitness for a particular purpose or noninfringement. In no event will CA be liable to the end user or any third party for any loss or damage, direct or indirect, from the use of this documentation, including without limitation, lost profits, business interruption, goodwill, or lost data, even if CA is expressly advised of such loss or damage.

The use of any product referenced in this documentation and this documentation is governed by the end user's applicable license agreement.

The manufacturer of this documentation is Computer Associates International, Inc.

Provided with "Restricted Rights" as set forth in 48 C.F.R. Section 12.212, 48 C.F.R. Sections 52.227-19(c)(1) and (2) or DFARS Section 252.227-7013(c)(1)(ii) or applicable successor provisions.

© 2002 Computer Associates International, Inc. (CA)

All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.

Contents

Chapter 1: Harvest Fundamentals

What Is Harvest?	1-1
Adaptability	1-2
Usability	1-2
Harvest Architecture	1-2
Harvest Objects	1-4
Projects	1-5
States	1-5
Processes	1-6
Repositories	1-6
Baseline	1-6
Working and Snapshot Views	1-7
Items	1-7
Versions	1-7
Packages	1-8
Package Groups	1-9
Forms	1-9
Users and User Groups	1-10
A Sample Life Cycle	1-11
Additional Information	1-12
Online Help	1-12
Contacting Technical Support	1-12

Chapter 2: Understanding Change Management

Client Directories and View Paths	2-1
Root Directory and Root View Path	2-2
Mirroring Internal and External Structure	2-2
Versions	2-5
Branching	2-5
Version Numbering	2-7

Merging	2-8
Version Tags	2-9
Merged Versions	2-9
Removed Item	2-9
Reserved Versions	2-10

Chapter 3: The Harvest Interface

Invoking Harvest	3-1
Changing Your Password	3-2
Changing the Broker	3-3
The Main Window	3-3
Menus	3-4
Toolbar	3-12
Taskbar	3-13
Workspace	3-16
List View	3-18
Output Log	3-18
Context	3-19
Kinds of Dialogs	3-21

Chapter 4: Single-User Agent

Starting a Single-User Agent	4-1
Starting a Single-User Agent on Windows Systems	4-2
Starting a Single-User Agent on Unix Systems	4-4
Connecting to an Agent	4-6
Using the Agent	4-7
Workbench	4-7
Remote Agent Connections From the Command-Line	4-8
Agent Shutdown	4-8

Chapter 5: Packages and Forms

Packages	5-1
Package Properties Dialog	5-2
Package Groups	5-6
Package Group Properties Dialog	5-7
Forms	5-8
Default Form Types	5-10
Concurrent Update	5-10

Form Attachments	5-13
Package and Form Associations	5-16
Add New Form Dialog	5-17
Package Movement Through the Life Cycle	5-19

Chapter 6: Processes

Approve	6-1
Check In	6-3
Checking In New Items	6-3
File Conversion	6-3
File and Item Case-Sensitivity	6-4
Signature Files	6-4
Package Check In	6-8
Command Line Check In	6-9
Check Out	6-11
Signature Files	6-11
Rules	6-11
File Permissions	6-12
File Conversion	6-13
File and Item Case-Sensitivity	6-14
File Date and Time	6-14
MVS Check Out	6-18
Command Line Check Out	6-19
Compare Views	6-21
Visual Difference	6-23
Concurrent Merge	6-25
Create Package	6-28
Cross Project Merge	6-29
Delete Version	6-34
Demote	6-35
Interactive Merge	6-37
Navigation	6-39
View	6-39
Resolving Conflicts	6-40
List Version	6-41
Examples	6-42
Move Package	6-44
Notify	6-46
Promote	6-48
Remove Item	6-51
Rename Item	6-52

Take Snapshot	6-53
User-Defined Process	6-56

Chapter 7: The Find Utility

Find File	7-2
Find Form	7-4
Find Package	7-8
Find or Select Version	7-12
Using Multiple Filtering Criteria	7-17
Select a Directory Path	7-21
Select Packages	7-22
Select Packages (Cross Project)	7-22
Select a Repository Item Path	7-23
Select User	7-24
Select a View	7-26

Chapter 8: ISPF Client Interface

Log In	8-1
Harvest ISPF Primary Panel	8-2
Settings	8-3
Harvest Context	8-3
Setting Context	8-3
Setting the Project	8-4
Setting the State	8-5
Setting the Package	8-6
Setting the View	8-7
Check In Process Selection	8-8
Check Out Process Selection	8-9
Check In	8-9
The Check In Panel	8-10
Check Out	8-12
The Check Out Panels	8-12
Integrating with ISPF Data Set List Utility (ISPF 3.4)	8-15
Mapping	8-15

Chapter 9: Using the Windows Extension (HarWind)

Log In	9-1
Set Default Context	9-3

Executing Check Out and Check In	9-6
Check Out Dialog	9-8
Check In Dialog	9-11
The Message Log	9-14
Log Out	9-14

Index

Harvest Fundamentals

The use of AllFusion Harvest Change Manager (referred to simply as Harvest) is divided into administrative tasks and day-to-day operational tasks.

Administrative tasks include defining the software development life cycle to use, setting up user groups, and initializing the physical repository.

The day-to-day operational tasks are described in this guide. These tasks include using the Harvest interface to manage change packages and execute processes in the development life cycle. See the *Administrator Guide* for a complete description of the Harvest administrative tasks.

What Is Harvest?

Today's development teams build large, distributed application systems. They work from heterogeneous platforms at remote locations and make simultaneous changes to a multitude of interrelated software modules and system documentation. The only way to effectively track this complex, enterprise-wide development activity is with a comprehensive, repository-based change and configuration management (CCM) solution. Manual methods and simple file-control systems are not robust enough to help you improve delivery and bolster service levels.

Harvest helps you synchronize development team activities on heterogeneous platforms, across your enterprise and throughout the entire application development life cycle. Harvest scales up to serve project teams working on your largest client/server enterprise systems and scales down to meet the needs of your smallest development teams.

The client and server portions can both be executed on the same machine or be distributed across multiple platforms. The client portion of Harvest consists of the graphical user interface (GUI) and the command line interface. The server portion contains most of the program logic.

Harvest's open architecture allows easy access to CCM information. Rather than developing yet another database standard, control information is stored within any of a number of commercially available relational databases. Database information is normally accessed from the GUI, but the database can be accessed directly to generate reports or integrate with other development tools.

Adaptability

One of the most powerful features of Harvest is its adaptability. An organization can define and follow a software development life cycle within Harvest, modeling it to the customary workflow. An organization can also integrate its own tools and processes into Harvest. Furthermore, Harvest enables development, maintenance, test, and problem-tracking activities to be synchronized across all platforms.

Usability

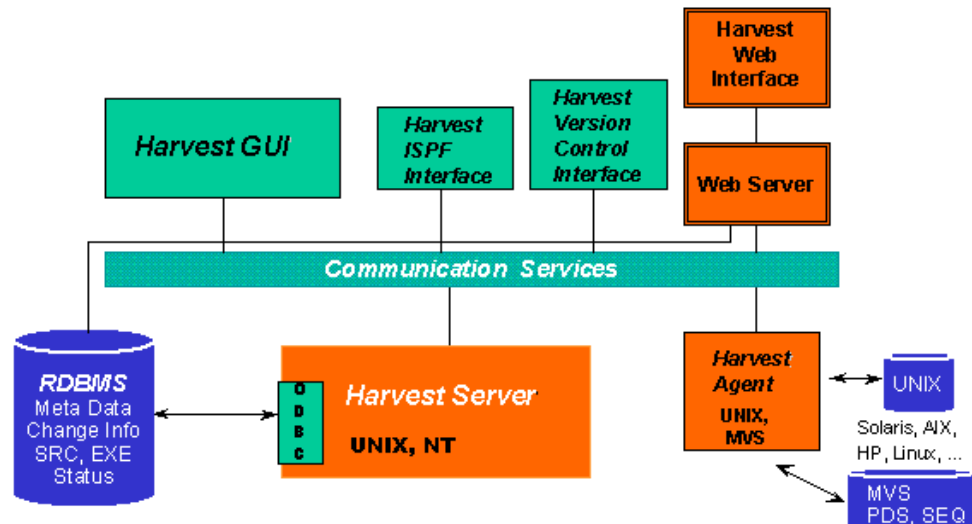
The Harvest user interface was built using state-of-the-art graphical user interface (GUI) technology. It was designed for ease-of-use and minimal training time. Standardized icons and toolbars are used throughout the interface. As an alternative to using a mouse, keyboard equivalents to toolbar menu items are provided.

Harvest has an object-oriented design. Objects are represented as icons or selection lists. Associated with objects are actions. This approach allows users to apply multiple unrelated actions to individual objects or groups of objects from a single location. It also means that the user never has to remember the name of an object.

Harvest Architecture

Harvest is a client/server application built to support distributed development and take advantage of the strengths of various platforms. The client/server model used by Harvest is called the *application server* model. In this model, the client process presents data and manages keyboard and device input. The application logic is defined and processed remotely by a group of dedicated application servers. The application servers, in turn, provide access to database servers.

The following figure illustrates the various components in the Harvest architecture.



The Harvest *client* for PC platforms consists of the graphical user interface (GUI) and command line interface. MVS functions can be executed from the ISPF client interface or the REXX command line interface. The client is the program that handles the user interface. It receives input from the user and sends it to the server for processing. It also presents the returned information to the user.

The *server* program executes operations requested by the Harvest clients. It contains the Harvest business logic.

The *ODBC layer* enables communication between the server and a database. The database is used to store all information under Harvest's control.

A *communication layer* provides the connection between the Harvest client and server. The communication layer consists of a set of protocols that allow the Harvest components to work together.

Part of the communication layer is a program called the *broker*. Because many clients and servers exist in one network, the broker assigns each client an appropriate server. Each server registers with the broker and provides the broker with enough information so that the client can find the appropriate server when requesting service. Each client then requests the broker to connect to an available server.

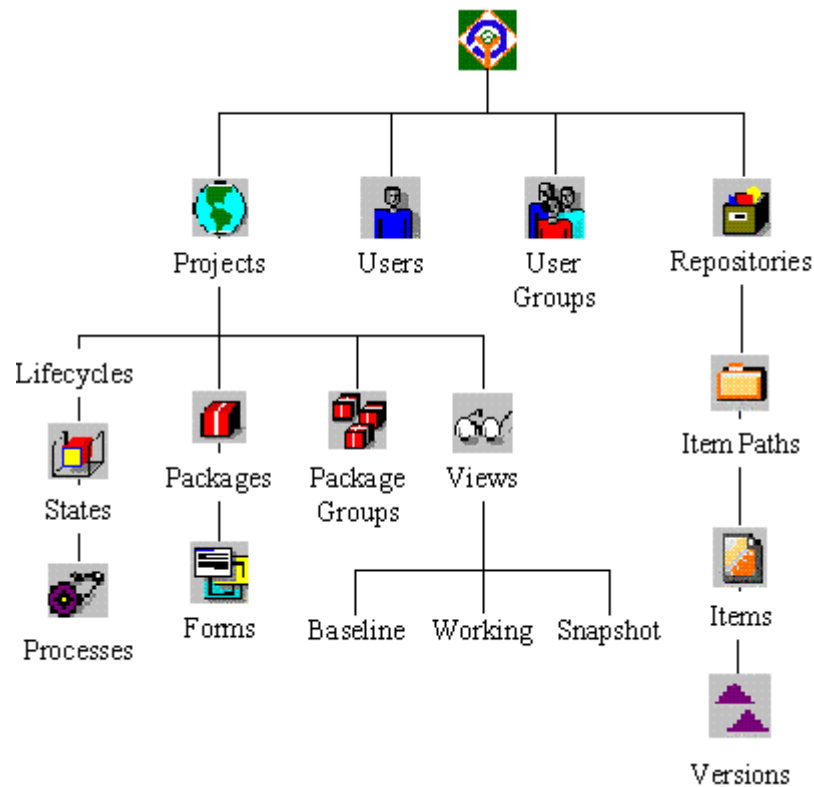
For users to successfully invoke the Harvest client, the following conditions must be met:

- The database processes must be running.
- The Harvest broker must be running.
- At least one Harvest server process must be running.

Harvest Objects

Complex CCM solutions can be built using a small number of basic object types. Key objects in Harvest include repositories, items, versions, projects, states, views, processes, users, user groups, packages, package groups, and forms.

The following figure illustrates the hierarchy of the various objects in Harvest.



Each Harvest object is briefly introduced in the sections that follow. More in-depth discussions are found in the appropriate sections of this guide.

Projects

The term *project* refers to the entire control framework within Harvest that supports a particular development or maintenance activity. A project includes information about how changes progress through the development life cycle, the activities that can occur, what data to access, and user responsibilities.

Note: In previous releases of Harvest, the term *environment* was used for *project*; they are synonymous.

Projects consist of the following objects:

- States
- Processes
- Packages and package groups
- Views
- Repositories
- Items and versions

You can have many different projects, depending on the applications being controlled and the kind of development activity undertaken. For example, you might have one project for maintaining an already released version of an application, another for developing the next release, and a third for maintaining code shared among various applications. You might have an entirely different project that the support group uses for tracking incidents and problems.

States

States are distinct work areas in which certain activities can take place as packages move from identification to completion. A life cycle can include any number of states.

A state is usually associated with a view. The view defines the scope of what the user can access in the state. A state can be defined without a view, if access to data items is not needed. For example, an initial state called Open, which is simply a holding area for newly created change requests, would not need a view.

Processes

A *process* is an action that can be executed on an object in Harvest. The processes defined for a state determine what activities can be performed in that state. Harvest includes a predefined set of process types. A set of valid processes can be defined for each state in the life cycle. A process name can be specified for each process defined to add clarity and to distinguish between multiple processes of the same type. When a process is added to a state, its name appears on that state's Processes menu.

Two processes determine the order in which change packages progress through a life cycle: *promote* and *demote*. A promote process moves change packages from one state to the next. A demote process returns change packages to a previous state.

Repositories

Harvest maintains the data under its control in one or more *repositories* stored in the RDB. You can put one application or several in a single repository or you can split an application over several repositories. A repository is independent of any particular project and is the basis for creating *views*, discussed later. Views can point to items in more than one repository, so it is not disadvantageous to use multiple repositories.

The term *item* refers to a component of a repository to distinguish it from a file in the external directory structure. To clarify the distinction between data outside the repository and data within it, a further convention is used. External directories are referred to as *client directories*. Directories within the repository are referred to as *item paths*, or *view paths* if a particular view is being referenced.

Baseline

Each project has a *baseline* that determines the collection of versions that can be accessed from within the project. The baseline can include items from one or more repositories and can include one or more snapshot views.

Working and Snapshot Views

Working and snapshot views are created based on the baseline. Each view looks at the same paths and items as the baseline, but not necessarily the same versions of items.

- Working views isolate functional work areas. These views are integrated into a life cycle by associating them with one or more states. Several states can share the same view, but a state cannot use more than one working view. The view determines which item versions are accessible to users. Changes introduced by a package are only visible within the package's current view, determined by the state it is in. As a package moves through a life cycle, the versions associated with it become visible in another view only when it reaches a state in that view.
- A snapshot view is a read-only image of a working view **at a specific point in time**. Snapshots enable users to capture a software inventory at significant points in its development, such as a release. Once a snapshot is created, it can be used to support other application management functions such as baselining.

A state can be associated with all snapshot views created in its project. This is accomplished by selecting the All Snapshot Views option in the State Properties dialog. When this option is selected, the state has as its view only the snapshots that are created from that project. The versions captured by snapshots can be checked out, but snapshot views are read-only, so items cannot be checked in.

Items

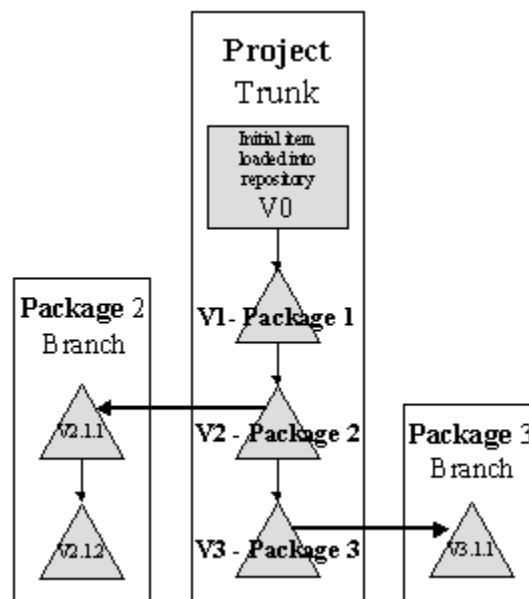
Repositories are composed of *items*. An item in the repository contains the actual data under Harvest's control. It is equivalent to a file on the filesystem. If you check out an item named foo.c from the repository, a file named foo.c is created on your filesystem.

Versions

Each item in a repository consists of *versions* of the item. Item versions are visible to users only through views; they cannot be seen by looking directly at the repository. *Base versions*, which are the initial versions of the items in a repository, are shown through both the baseline and all working views. Subsequent versions created after the baseline has been established are shown through working views only.

When a user checks in a new version of an item, that new version becomes visible only from within the states that use the same view as the state the user is checking in from. For example, if a user checks in version 2 of file1.c from the Coding state, and the Unit Test state shares the same view as the Coding state, version 2 of file1.c is visible in both states. However, version 2 is invisible in the Release state because that state does not share the same view. In order for version 2 to show up in the Release state, the package that was used to create the version must be promoted to the Release state.

Versions are managed in Harvest in a tree structure called a *delta tree*. The following figure depicts a single item that has been updated in two projects: Release 1.0 and Release 2.0. The triangle Δ represents a *delta*, or change, made to the original item as loaded in the repository. Each delta represents a new version.



The original item is stored in the repository. Projects use this repository item to create versions on *trunks* of the delta tree that extend from the repository. These trunks can support versions on multiple *branches*. Branches are independent delta chains that extend from versions on the project trunks.

Packages

A *package* is the basic unit of work that moves through a life cycle. It typically represents a problem or a request that needs to be tracked, the changes made in response to the problem or incident, and any other associated information. The package is the user's link to the actual data accessed during the change process. Each package resides within a particular state of the life cycle.

Packages can be associated with one or more *forms*, which are used to organize and maintain related information. The package/form association is a powerful way to combine problem tracking with change control. (Packages can be created without an associated form.)

The life span of a package usually takes place within one project; however, packages can be created in one project and then moved to another to begin a new life cycle. This use of packages allows a problem tracking project to be closely linked with a change control project. For example, problems reported to a customer support group can be tracked by packages in the support project that pass through several stages. Eventually a problem is assigned. Instead of beginning a new package in the development project, the package with its history intact can be moved from the support project.

Package Groups

A typical project can have many packages. A *package group* provides a higher logical level for operating on related packages. A package can belong to one group or several, or it might not belong to any.

Any operation that can be performed on a package can be performed on all packages in a package group. For example, a user can promote all packages in a group from one state to the next. If an approval is required before packages can move from one state to another, the entire group can be approved.

Package groups also provide support for filtering operations and allow you to easily set up variant events for special kinds of packages. For example, packages in a certain group might need to pass through an extra state before continuing through the life cycle. This can be the case with documentation changes, which go through a special documentation state before moving on through the normal quality assurance (QA) and release cycles.

The *bind packages* feature ensures that packages assigned to a group can move through the life cycle as a group. Packages can still move individually through the life cycle if the Enforce Bind is not enabled. If a package is part of a bound package group and if Enforce Bind is enabled on the Demote or Promote Properties dialog, all packages in the group must be demoted or promoted together.

Forms

Harvest *forms* can be used in a similar way as paper forms are used. For example, they can be used to track issues and problems or as a structured method of communication.

Forms are always associated with packages. This association provides the link between the problem tracking and change tracking functions in Harvest. All the information gathered through a form is also directly accessible from the associated package.

Users and User Groups

Users are individuals defined in Harvest. When Harvest is installed, an initial user is automatically created (see the *Install Guide*). This user must add other users who are involved in the setup activities.

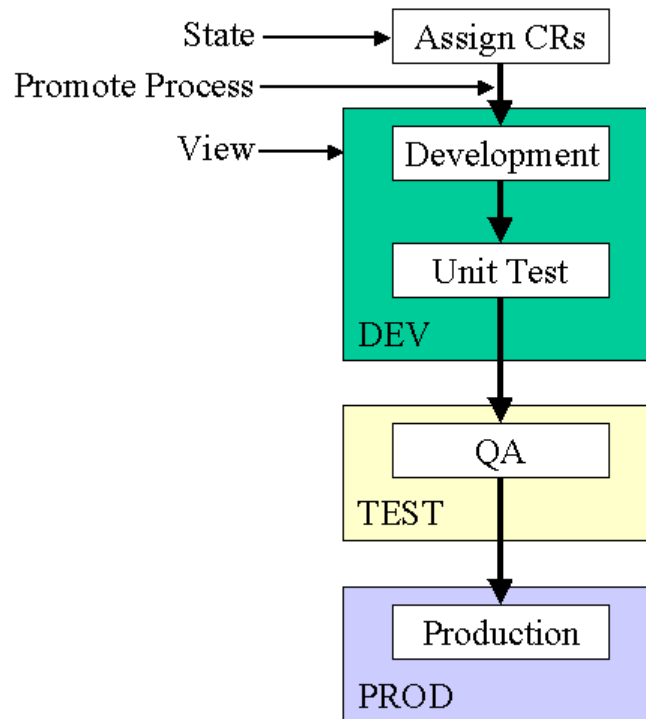
A *user group* is a Harvest object that allows users to be arranged in groups for the purpose of access, approval or notification. Two predefined user groups are supplied with Harvest called Public and Administrator.

Users and user groups exist at the global level within Harvest, which means that they are available in all projects defined within a Harvest installation. A user can belong to any number of user groups, and the groups imply no hierarchy. For example, a user in the Development Manager group does not implicitly belong to the Developer group.

Creating users and user groups are closely related tasks. If you create users first, you can add them to the groups when you define the groups. If you create groups first, you can add the users to them when you define the users.

A Sample Life Cycle

Harvest enables you to create a life cycle that suits your application development procedures. The following figure illustrates some of the basic building blocks used to create a Harvest life cycle.



In this example five states exist. Assign CRs is the state in which packages are created to fix change requests. Development is the state in which developers modify code, and Unit Test is a state in which they test their changes. QA is the state in which QA personnel test the application and Production is the state that holds code ready to be released.

Note that state names are entirely user-definable. The Assign CRs state is based on the terminology of change requests. If you are accustomed to creating trouble reports rather than change requests, your initial state might be Assign TRs.

The black arrows indicate promote processes that define how packages move from one state to another. The states and promote processes combine to form a road map through which an application "travels" on its way to release. The vehicle by which it travels is a package. A package typically represents a change request (CR) for the application.

In the model depicted in this figure, a package starts in the Assign CRs state and is promoted to Development. A view is not associated with the Assign CRs state because no activities in this state affect actual data.

Developers accessing a package in the Development state can modify versions of items within the DEV view. The changes made in DEV are not visible in any other view until the package is promoted to a state in that view. In this example, after a package has completed testing in Unit Test, it is promoted to QA, which is in the TEST view. When the package enters QA, the TEST view is updated with the package changes. Changes made in the DEV view by other packages do not appear in TEST until those packages are promoted to QA.

Additional Information

For additional assistance, refer to Harvest's online help and user documentation, or contact Technical Support at Computer Associates International, Inc.

Online Help

Harvest provides a comprehensive online help facility. The online help contains information about dialogs and fields to help you perform specific tasks in Harvest. You can access online help for Harvest at any time by selecting either the Help menu bar item or a Help button.

Contacting Technical Support

You can contact us with any questions or problems you have. For technical support, visit the website: <http://esupport.ca.com>.

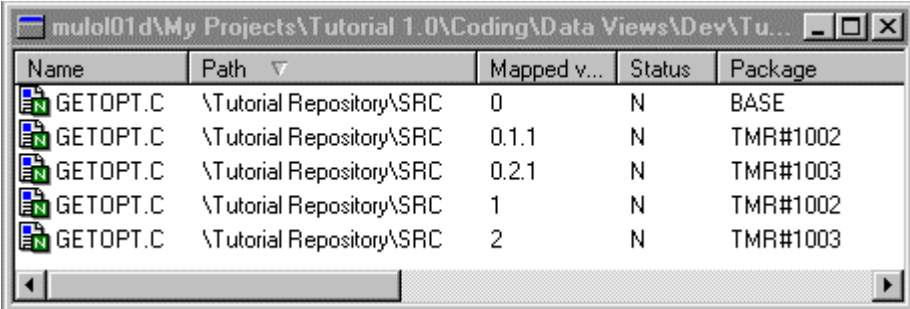
Understanding Change Management

Managing changes is the heart of any CCM system. Harvest extends CCM functionality beyond change management by providing problem tracking, life cycle modeling, process management, and package management. Change management, however, remains a central and indispensable part of the Harvest system. This chapter explains how changes are managed in Harvest.

Client Directories and View Paths

As you move about in the client directories or in the repositories, your path is defined by your current position in the hierarchy. A *view path* is a location within the Harvest repository. The project and state you select define what view paths and versions you can see.

Double-clicking an item displays its versions in the list view. You can browse a version by double-clicking it to open the version in the list view. The following figure shows the list view, displaying the versions of an item selected under Data Views on the Projects tab.



Name	Path	Mapped v...	Status	Package
GETOPT.C	\Tutorial Repository\SRC	0	N	BASE
GETOPT.C	\Tutorial Repository\SRC	0.1.1	N	TMR#1002
GETOPT.C	\Tutorial Repository\SRC	0.2.1	N	TMR#1003
GETOPT.C	\Tutorial Repository\SRC	1	N	TMR#1002
GETOPT.C	\Tutorial Repository\SRC	2	N	TMR#1003

Root Directory and Root View Path

Harvest uses the concept of a client root directory and a view path root to allow you to control how your client directory path and view path change. In the Find Version dialog and in the check in and check out process execution dialogs, you can set the client root directory or view path root to a certain location. This allows you to synchronize the check in and check out processes with the client directory and view path structure already established.

Two distinct roots can be defined:

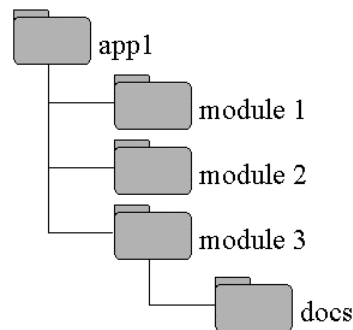
- *Client Root Directory* when viewing files. By selecting your client root directory, you are setting a default client directory that is inserted in the client root directory field during subsequent check in processes.
- *View Path Root* for repository paths, when viewing items and versions. By selecting your view path root, you are setting a default view path that is inserted in the view path root field during subsequent check out processes.

Mirroring Internal and External Structure

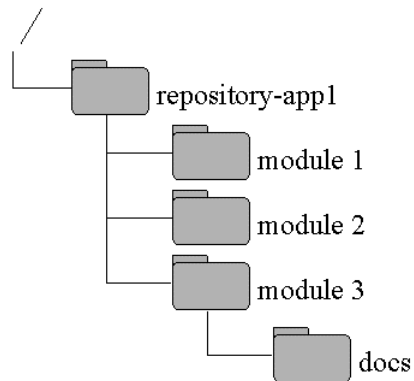
When a repository is created, an administrator loads a set of directories and files into it. The external directory structure is duplicated in the repository and in the baseline of any projects that use it. When viewing the paths in a repository, the top-level of the path is the repository name, and the application directory structure lies beneath it.

Typically, a directory structure similar to the repository structure is maintained on the client by individual developers, and is used for updating application items and building the application for testing.

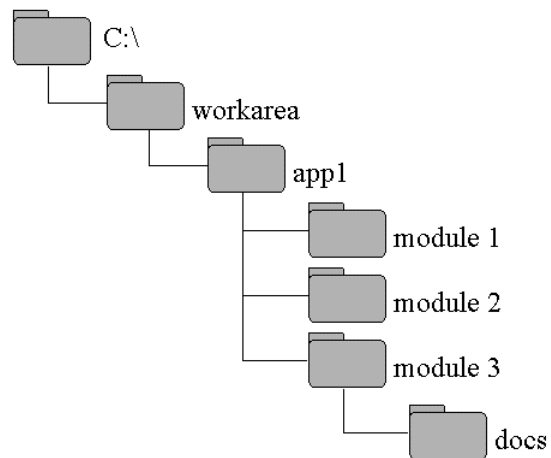
For example, the following figure illustrates the client directory structure of an application called app1:



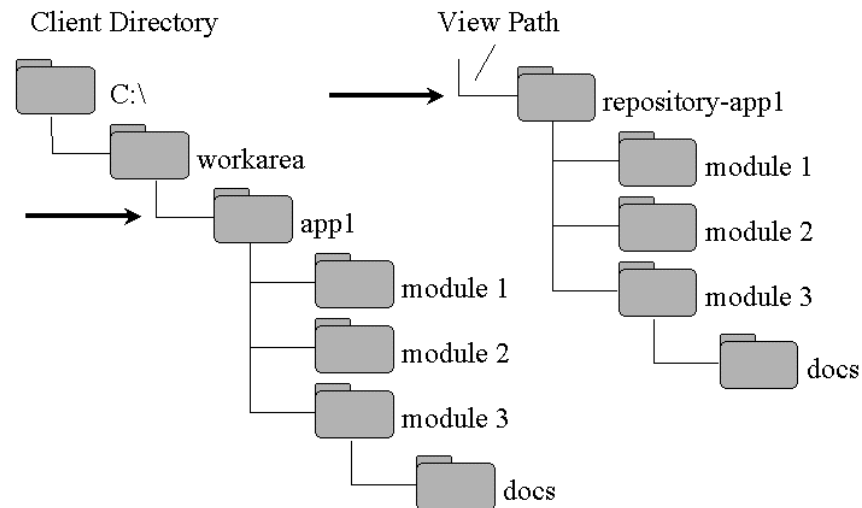
When this application is loaded into a repository named repository-app1, the application structure is duplicated in the repository. When users access Harvest to look at items in a view that includes this repository, the repository paths look like the following illustration.



Assume that you have been working on this application and it has working directories on a Windows 95 client mirroring the application structure. Your work area looks like the following:



You can use roots to synchronize the internal and external directory structure. In this case, you should set the view path root to `/repository-app1` and the client directory root to `c:\workarea\app1`. These two roots mark the point of synchronization between items inside the repository and their corresponding files in the client directory.



Directory/Path Synchronization

The recursive search operation in Harvest is used to synchronize a client directory and a view path. During a recursive search operation, Harvest starts at a specified client directory or view path and searches all directories or paths below it for files or items to operate on.

The recursive search option is typically used with other options available on the check in and check out windows that let you specify how the synchronization should occur. Harvest provides three choices:

- You can preserve the directory or path structure. This causes Harvest to look for a matching path for each directory.
- You can preserve and create the directory or path structure. Harvest matches paths and directories and creates corresponding ones if they are missing.
- You can take items from multiple paths or files from multiple directories and place them all in one directory or path, disregarding the structure.

Versions

When a version is created, Harvest maintains extensive information about the change. Some of the change details retained by Harvest are:

- Name of the user making the change.
- Date and time the change was made.
- Name of the package associated with the change.
- A change description entered at the time the update was made.
- A tag indicating if the version is the result of a merge, check out, or remove item process.
- Information about the host file checked in to create the delta, including the file modification time stamp, file size, and user access.

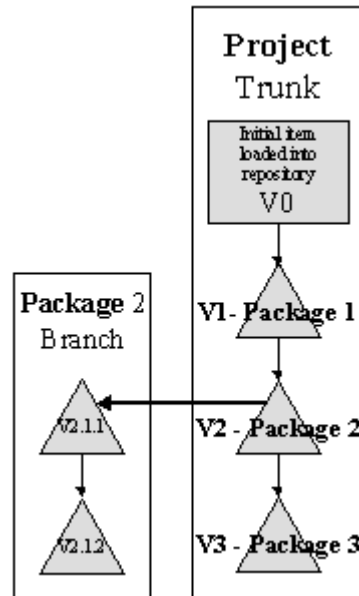
Branching

Harvest maintains only one copy of an item; however, often the same item needs to be updated by different users at the same time. Harvest supports this kind of *concurrent development* by allowing changes to be made on branches off the project trunk. These branches are independent delta chains that extend from trunk versions. Branch versions are not affected by any changes made to the same item, either in the project's trunk or on other branches.

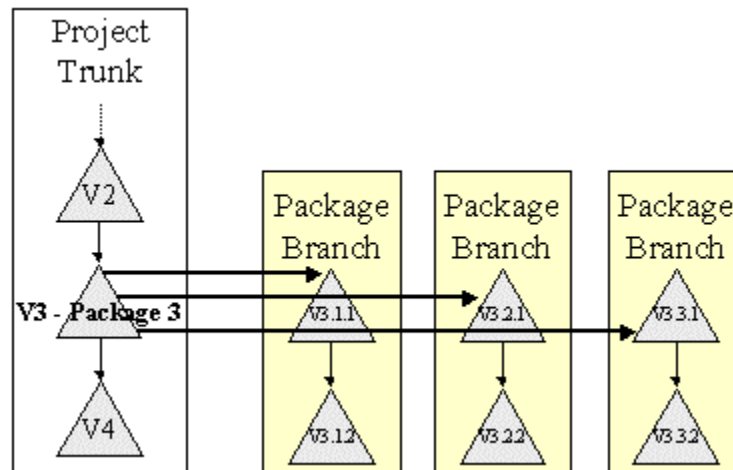
When the Concurrent Update mode is specified during a check out process, a reserved version is created. This reserved version acts as a placeholder for a new branch. Modifying the checked out file and then checking it in, creates a new branch version. Until the package containing the branch version is merged back to the project trunk, all changes made to that version remain on its branch.

Note: Branch versions are only visible in the view in which their packages are currently located or in the views that the package has traversed.

In the following example, version 2 of an item is checked out for Concurrent Update. When the item is modified and checked in, a branch version 2.1.1 is created. Checking out version 2.1.1, modifying it and checking it back in creates a version 2.1.2 that is also located on the branch.



Multiple branches can be created off of the same version on the trunk, and any trunk version can be checked out to a branch, not just the latest. In the above project, checking out version 3 for Concurrent Update a second time, modifying it, and checking it in creates a version 3.2.1 on a second branch. A third check out of version 3 for Concurrent Update, and subsequent check in, creates a version 3.3.1 on a third branch. Meanwhile, version 3 on the trunk can be updated to create a new version 4.



Note that the presence of a new version on the trunk does not affect branches, which can be taken from any trunk version and which remain completely independent from changes on the trunk. The newest trunk version is not affected by the existence of branches, which do not belong to a view. Until a branch is merged, the delta chain on the trunk and deltas chains on branches are completely independent of each other.

Initial branch versions can only be made from trunk versions. Versions located on a branch can be checked out for Update or Concurrent Update; both modes of check out create a reserved version on the same branch. Checking out a branch version for Concurrent Update does not create a new branch from the existing branch.

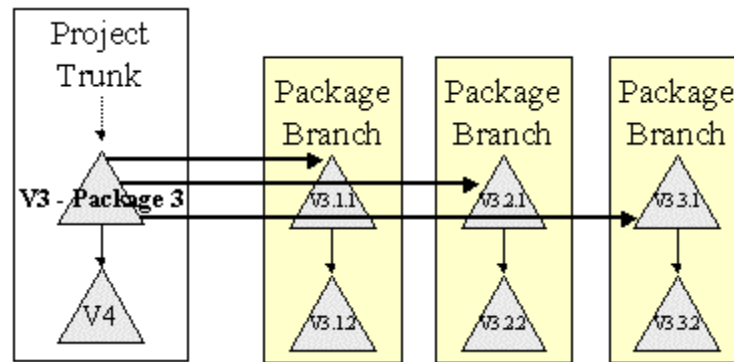
These rules apply to branch versions and their packages:

- Branch versions are always associated with specific packages. Only the number of packages available limits the number of branches that can be made from a single trunk version.
- A package can contain only one reserved version of an item.
- A package cannot be used to check out an item using the Concurrent Update mode if the package has a reserved version of the same item.
- If a package contains a branch version, and the package is moved to a different view, only the users who have access to that view have access to the branch version.

Version Numbering

The convention adopted by Harvest for version numbering is similar to that used by other version control products that support branching. The first digit of a branch version number refers to the trunk version the branch was created from, the second to the branch number and the third to the version sequence. For example, version 3.2.1 refers to the first version on the second branch from trunk version 3.

In the following example, version 3 of an item has been checked out for Concurrent Update three times. When the files are modified and checked in, branch versions 3.1.1, 3.2.1, and 3.3.1 are created. When changes are made to version 3.1.1 on the first branch, and the changes are checked in, the new version created on the branch is 3.1.2.



Merging

Branching in Harvest is resolved through merge processes. Merge processes enable you to view changes made to two different versions of an item, combine changes, and discard changes.

Merges are typically accomplished in two stages, and can be used to merge versions in the same project or across projects.

- Merging a branch version, created by the Concurrent Update mode of check out, to the project trunk requires the concurrent merge process.
- Merging across projects requires use of the cross project merge process.
- The interactive merge process is the second stage required for both types of merges. The interactive merge process can also be used as a single step for merging branch versions for a specific item.

See the chapter "Processes" for details about each type of merge.

Version Tags

Version tags identify the status of items and versions. These different tags denote specific versions:

- Merged (M)
- Removed, or logically deleted (D)
- Reserved (R)

Merged Versions

Branch versions must be merged to the trunk before their changes are included in the project.

- Executing the concurrent merge process on the package associated with the branch version creates a new version on the trunk.
- Executing the cross project merge process will create new versions in the destination project.

The versions resulting from the merge processes are the latest on the project's trunk and in any view of the project. If conflicting changes exist, the new versions are marked as merged versions. Merged versions are marked with an M, and have version information associated with them, including package, user name, date, and time.

These rules apply to merged versions:

- Only the interactive merge and the delete version processes can be performed on a merged version.
- Promoting or demoting packages that contain a merged version to a state in another view is not allowed.
- Another version of the merged version cannot be created; the merged version must first be resolved through the interactive merge process. The merge tag is then removed.

Removed Item

The remove item process enables you to delete items from the current view. The item is not gone; the remove item process creates a new version of the item that is marked as removed with a D tag. The removed item has attributes similar to other versions, and can be seen in version view or through the Find Version dialog but it is no longer displayed in item view.

These considerations apply to removed items:

- The removed item is associated with its package and can progress through the life cycle.
- To restore an item that has been removed, the delete version process can be used to delete the removed version.
- If the latest version of an item in a view is a removed item, and a snapshot is made from the view, no versions of the item are captured by the snapshot. Therefore, if a project's baseline is based on the snapshot, no versions of the item exist in that project.

Reserved Versions

To check out an item, it must first be reserved by a package. When you check out an item using the Update mode or the Reserve Only mode, a version placeholder, or reserved version, is created for the associated package.

- The Update mode copies the selected versions to the destination client directory and creates a reserved version.
- The Reserve Only mode does not move any data to the client directory but creates a reserved version.

Reserved versions have version information associated with them, including package, user name, date, and time. Reserved versions are marked with an R, and are displayed in the Versions view and in the Find Version dialog.

These considerations apply to reserved versions:

- They cannot be checked out using the Update or Concurrent Update modes.
- If a reserved version is checked out for Browse, it has the same content as the previous version on which it is based.
- They can be included in report functions.

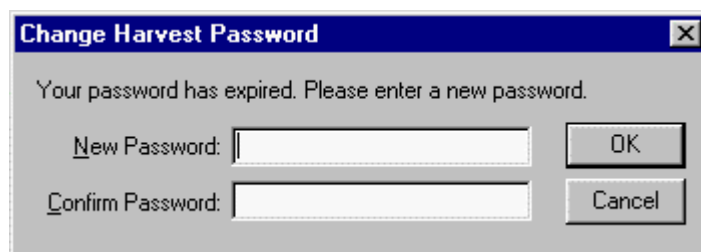
The Harvest Interface

This chapter provides you with the information you need to invoke Harvest and use the main window. The functions of the taskbar, workspace, list view, and output are explained.

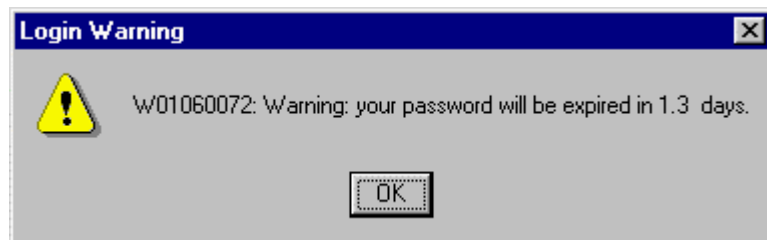
Invoking Harvest

When Harvest is first installed, an initial user is created. This user must add other users who will be accessing Harvest. The Harvest Administrator using the User Properties dialog defines a user's initial password. The Administrator may optionally set password policy to define rules for password validation.

If you attempt to log on with an expired password, Harvest will prompt for a different password.



If the password policy defines an expiration warning, Harvest will issue a warning message indicating how many days remain before password expiration. See the section "Changing Your Password" to change your password.



After logging in, the main window shows the *Administrator window* or the *workbench*, depending on your Harvest program selection. The Administrator window provides you with functionality suitable for administrators, whereas the workbench provides you with access to all user functions.

Windows

To log in to the Administrator window on a Windows 9x, NT or 2000 system:

1. Select Administrator from the program group.
2. Enter your user name, password, and the broker location in the Harvest login dialog, and click OK.

After logging in, the main window shows the Administrator window, which provides you with access to all administrative functions.

To log in to the workbench on a Windows 9x, NT or 2000 system:

1. Select Workbench from the program group.
2. Enter your user name, password, and the broker location in the Harvest login dialog, and click OK.
3. The Default Context dialog displays. You can set your context and click the OK button, or click Cancel; both actions open the workbench. (See the section "Context" for information about setting your context and the Default Context dialog.)

After logging in, the main window shows the workbench, which provides you with access to all user functions.

Harvest Web Interface

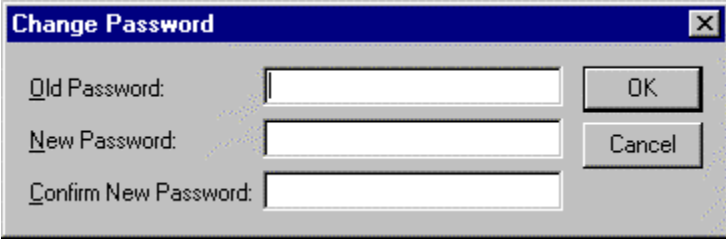
For information about logging in and the web interface functions (Harweb), see the Web Interface HTML help documentation.

ISPF

For information about logging in and the ISPF interface functions, see the chapter "The ISPF Client Interface."

Changing Your Password

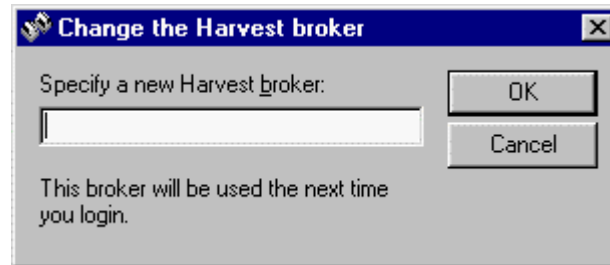
You can change your password. On the Administrator window or the workbench, choose Tools, Change Password to open the Change Password dialog in which you can change your password.



The image shows a Windows-style dialog box titled "Change Password". It has a blue title bar with a close button (X) in the top right corner. The dialog contains three text input fields: "Old Password:", "New Password:", and "Confirm New Password:". To the right of these fields are two buttons: "OK" and "Cancel".

Changing the Broker

You can change the broker to which you connect. Select Change Broker from the program group, in the dialog enter the broker location to which you want to connect, and click OK.



The Main Window

After logging in to Harvest, the main window displays. The main window depicts Harvest objects and their relationships and provides access to functions. The window is divided into four panes: the *taskbar* on the left, the *workspace* in the middle, the *list view* on the right, and the *output log* across the bottom. The *status bar* is the horizontal area at the bottom of the window and provides information about the current menu you have selected in the main window. When executing time-consuming operations, such as load repository, a progress status bar also displays.

- When logging in as an administrator, the main window is called the *Administrator window* and provides functionality suitable for an administrator.
- When logging in as a user, the main window is called the *workbench* and provides functionality suitable for a user.



You can change the name of objects in the workspace and in the list view by selecting the object name and then clicking it again. The name becomes editable and you can modify it, after editing press the Enter key. The new name is automatically updated in the workspace, the list view and the properties.

The following rules apply to the rename feature:

- You can only rename an object to which you have update access.
- You cannot rename folders.

Menus

The menu bar at the top of the screen contains menus such as File and View. Menus on the main window provide you with access to functions and processes. You can also right-click an object in the workspace or list view to invoke a shortcut menu, which includes available processes and the properties dialog for the object. Keyboard shortcuts appear on the menus to the right of many commands.

The File, View and Window menus provide conventional functionality such as Window, New Window. The unique Harvest menus are described below.

View

The View menu allows you to choose main window display options, such as showing the workspace and output log.

The Package Filter option enables you to show package folders according to the option you choose. The Package Filter options are:

My Packages—Shows all packages assigned to you.

All Packages—Shows all packages in current project.

After choosing an option, the package folder name in the workspace is refreshed to show your selection. For example, when you choose My Packages, the package folders are named My Packages and contain the packages assigned to you. If you then choose All Packages, the package folders are named Packages and contain all the packages in the current project.

Processes

The Processes menu lists all processes available for a selected state. This menu is disabled if a state has not been selected. The content of this menu varies for each state, depending on how the life cycle and processes have been defined. The order of the processes is alphabetical by process name. Choosing a process from this menu opens the associated process execution dialog.

Reports

Using the Reports menu, you are able to generate Harvest, project-related, and package-related reports specific to your current Harvest context. A variety of reports are listed, and you can choose a report name to select and generate the report. The Reports menu options differ on the Administrator window and the workbench. All reports display in the output log unformatted.

Administrator

The Administrator report options include Harvest-level and project-level reports. Initially all project-level report options are disabled, you must click on a project to activate the reports options. In addition, if you do not have access privileges the option is disabled. You can also access the project-level report options from project shortcut menu.

The Administrator window reports incorporate project management information from Harvest. These reports allow quick determination of projects' progress and provide metrics to access project and process activities within a project. Problem areas can be located quickly, to identify and isolate small problems before they become critical.

Level	Report Name	Description
Harvest	Project Summary	Lists detailed information about each Harvest project.
	Repository Summary	Lists detailed information about each Harvest repository.
	Harvest Access	Lists what access the specified user has to the Harvest objects.
	Users	Lists detailed information about each Harvest user.
	User Groups	Lists detailed information about each Harvest user group.
Project	Life Cycle Definition	Lists all process/process types for each state in the selected project.
	Project Access	Lists all project and state access belonging to a selected project.
	Approval Definition	Lists all approve processes with approval user/user group in a selected project.
	User List	Lists all users in each user group.

Workbench

The workbench report options include package-related reports. Initially the report options are disabled, you must click on a project to activate the reports options. In addition, if you do not have access privileges the option is disabled. You can also access the report options from project shortcut menu.

Level	Report Name	Description
Project	Package History	Lists the package history of all the packages in a project.
	Package Approval	Lists the history of package approvals and rejections in a project.

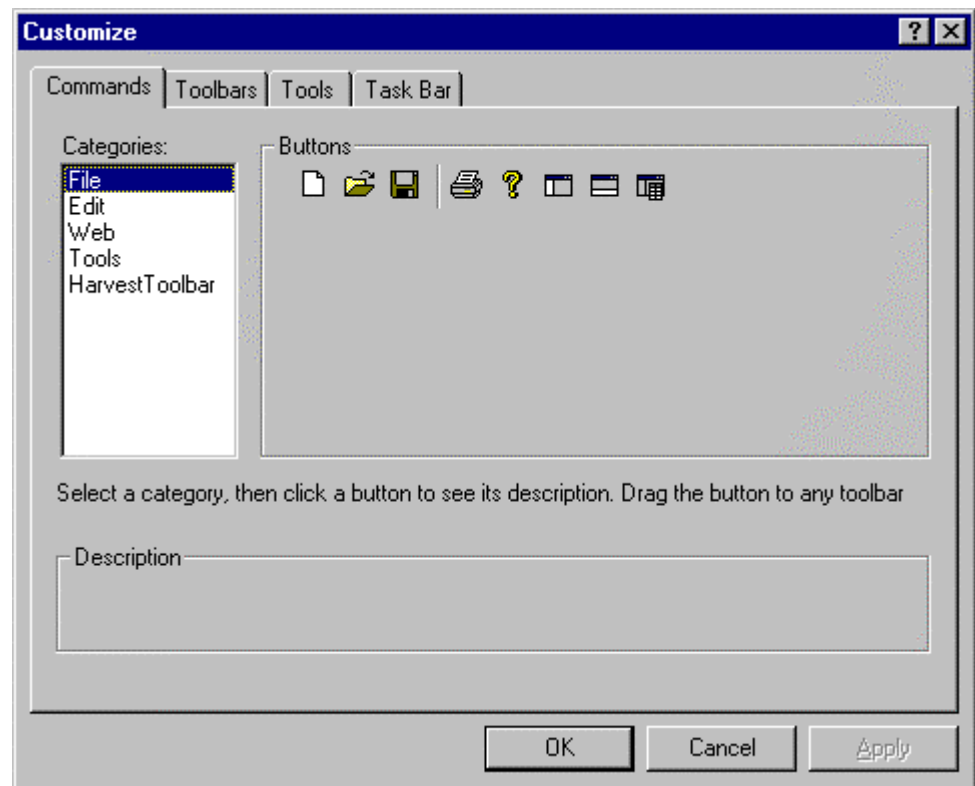
Tools

The Tools menu differs on the Administrator window and the workbench:

- On the Administrator window, the Tools menu enables you to customize your Harvest menus and toolbars, and to establish window and file default settings. Using the Tools menu you can also create new objects after selecting the object type, for example, select the States folder and then choose Tools, New, State to create a new state.
- On the workbench, the Tools menu enables you to customize your Harvest menus and toolbars, and to establish window and file default settings. Using the Tools menu you can also invoke the Find Version, Find Package, Find Form and Default Context dialogs.

Customize Dialog

The Customize option opens the Customize dialog that enables you to customize your Harvest menus, toolbars and the taskbar.

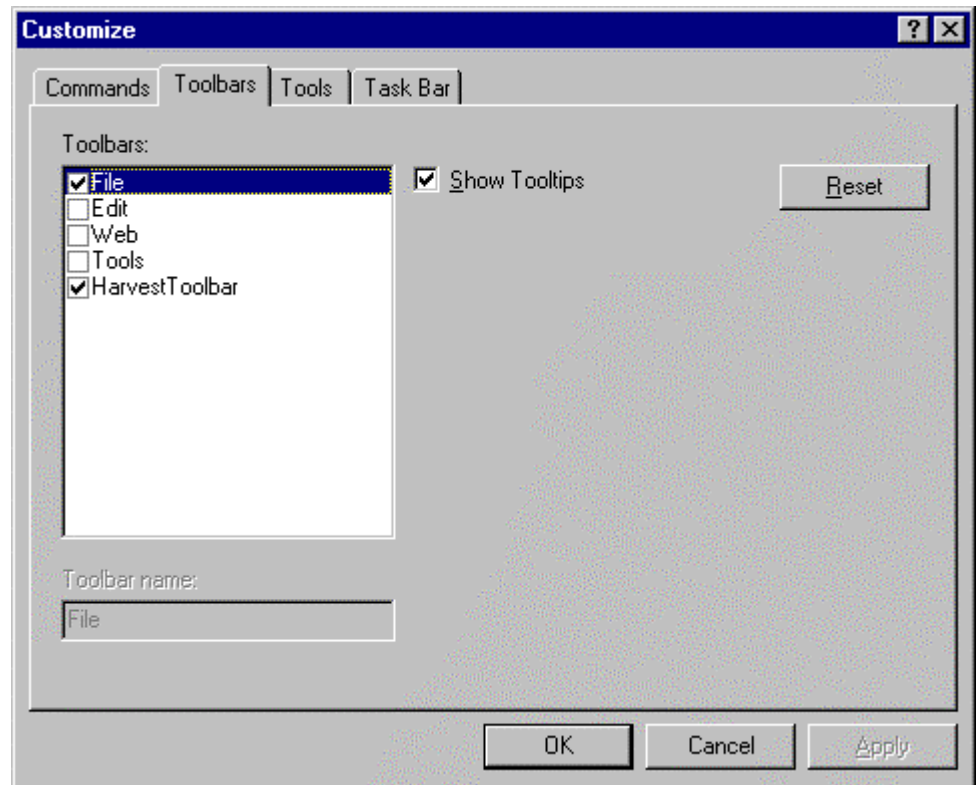


Commands

A menu lists commands. Some of these commands have images next to them so you can quickly associate the command with the corresponding image, making the command easily accessible.

To add an image to a command:

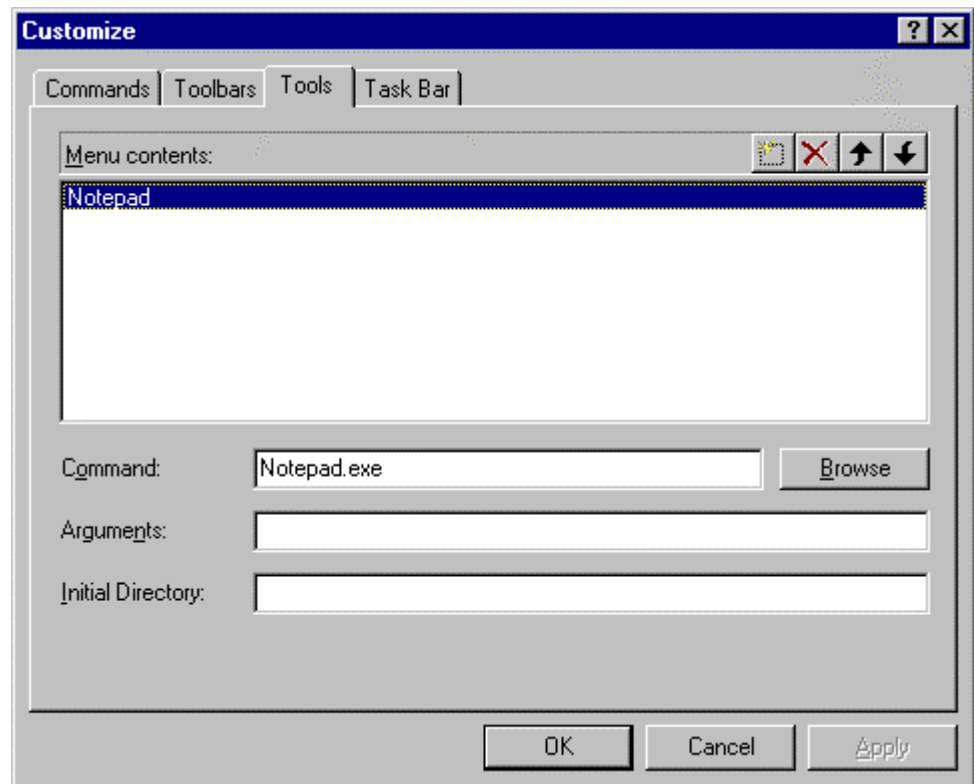
1. Select a category to display the buttons available.
2. Choose the button you want to add to a menu or toolbar.
3. Drag the button to the menu or toolbar to create a toolbar button for that command.



Toolbars

Toolbars allow you to organize the commands so you can find and use them quickly. You can easily customize toolbars; for example, you can add and remove menus and buttons, create your own custom toolbars, hide or display toolbars, and move toolbars. Toolbars can contain buttons, menus, or a combination of both.

Use the Toolbars tab to select which toolbars to hide or display. You can enable the Tooltips option by selecting the Show Tooltips check box.



Tools

You can add options to the Tools menu contents. This enables you to run external programs by choosing the option from the menu.

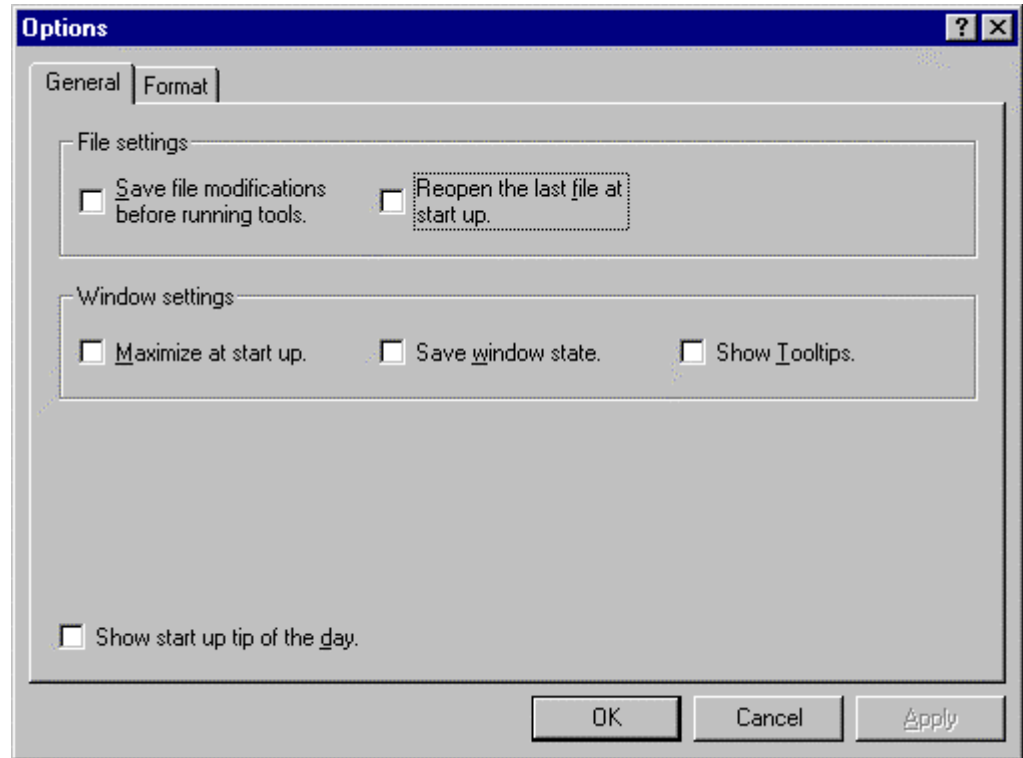
Command—Specify the name and location of the program you want to run.

Arguments—Enter any additional command line options in this field.

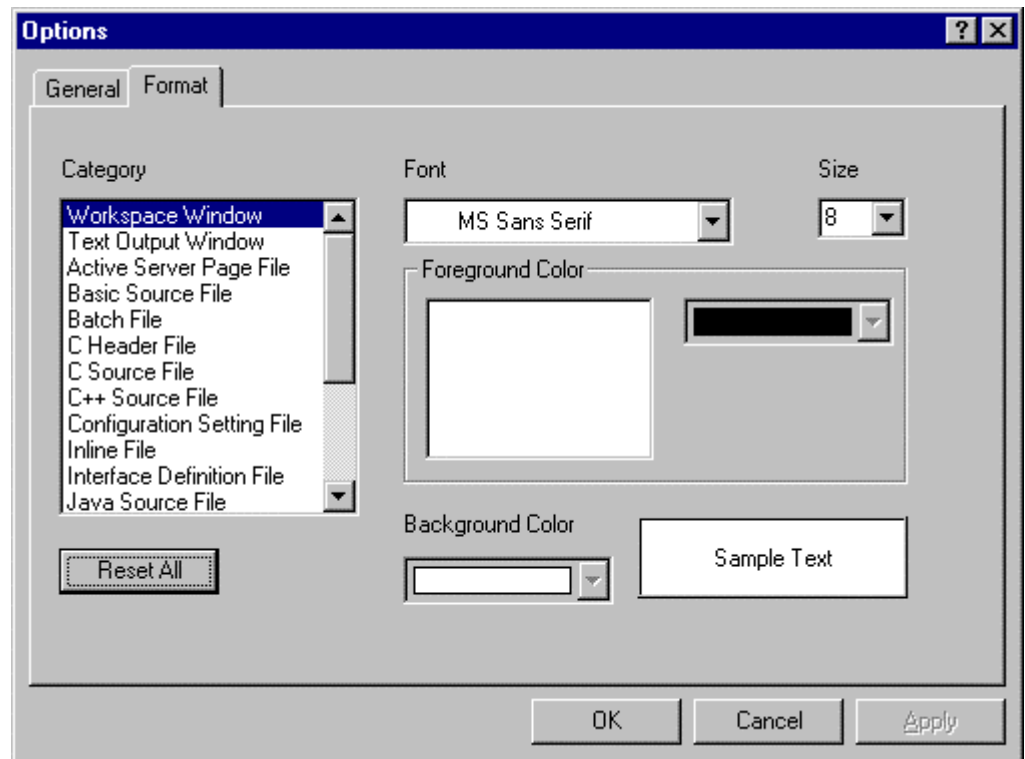
Initial Directory—Specify the directory that contains the data on which you want the program to operate.

Options

Choosing Options opens the Options dialog which allows you to set file and window settings.



Using the Format page you can set the type and size of font for your Harvest display, and choose foreground/background colors. The Sample Text box shows an example of your current settings.



Find

Using the Tools menu, you can open:

- The Find Form dialog to locate forms and their associated packages.
- The Find Package dialog to locate packages.
- The Find Version dialog to locate item versions in the Harvest database.

See the chapter "The Find Utility" for information about the Find dialogs.

Help

The Help menu allows you to access online help for Harvest. Once the help system is invoked, you can use the table of contents to find topics, or use keyword searches to locate a particular topic.

Toolbar

The toolbar at the top of the screen contains buttons that offer many of the menu options such as:

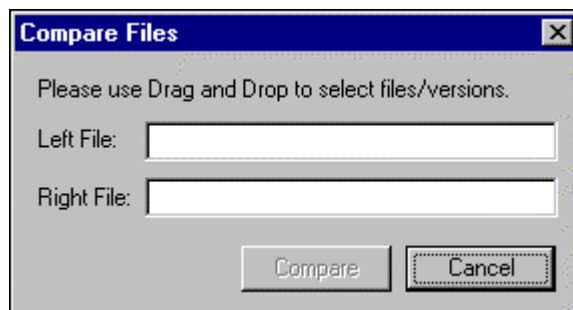
- The Find functions enable you to locate objects by specifying filtering criteria and to execute available processes on your selection, see the chapter "The Find Utility."
- The Set Context function.
- The Compare Files button that allows you to easily view differences between files or versions.
- The Customize Dialog button allows you to customize the toolbar.
- The GoUpOneLevel button allows you to navigate up the directory structure of the Data View.
- The Refresh Packages button refreshes the package list in the workspace, and the packages in the list view. Any package lists that are visible in the workspace are refreshed.

Compare Files

You can compare two files or versions by using the Compare Files toolbar button. This button enables you to open the Compare Files dialog and then generate a Visual Difference report. When the Visual Difference process is invoked, the versions being compared are displayed with common blocks (lines the same in both versions) and conflict blocks. Because the compare files process is read-only, changes cannot be made to the files.

To compare two files or versions:

1. On the workbench toolbar, click the Compare Files button to open the Compare Files dialog.



2. Select files on Windows Explorer or versions in the workspace.
3. Populate the fields in the Compare Files dialog by dragging files or versions into the fields.

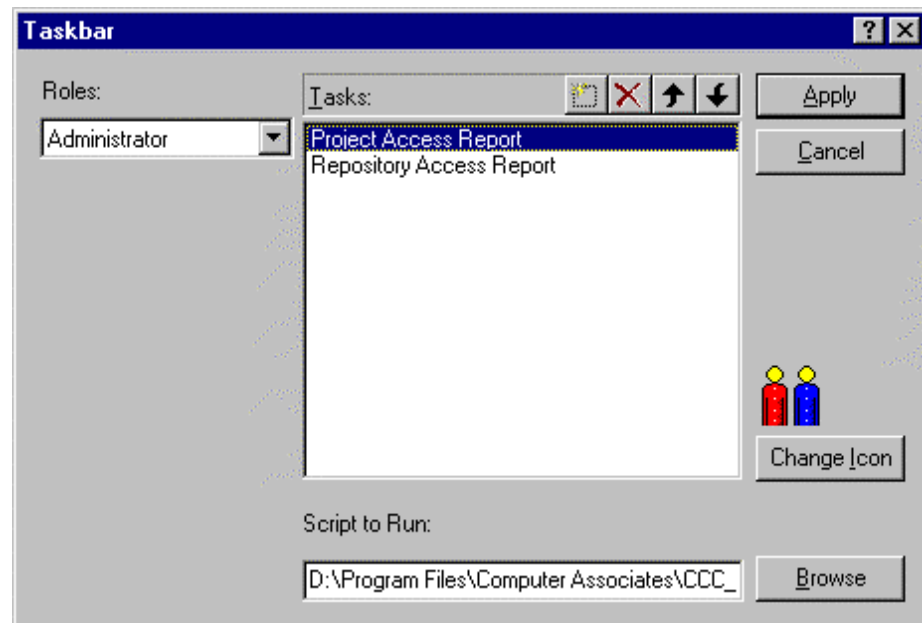
4. Click the Compare button to show a Visual Difference report. For more information about the Visual Difference report, see the chapter "Processes" in this guide.

Taskbar

The taskbar is the left pane on the main window. From the taskbar you can create shortcuts to launch and interpret Visual Basic and Java scripts. The taskbar can be divided into groups of shortcuts to help organize your information. You can add and remove groups, shortcuts, change the icons, and hide the taskbar.

- Click **once** on a shortcut on the taskbar to run a task without exiting the main window.
- Change groups to move to a different set of shortcuts.

You can set up the taskbar by using the Taskbar dialog. The Taskbar dialog is available from the shortcut menu on the taskbar or by choosing Tools, Customize and clicking the Taskbar tab.



Roles—The name of your current group is shown in this drop-down list. You can choose a different group by choosing its name from the list.

Using the Tasks field and toolbar, you can add tasks, delete tasks, and arrange the order of tasks in your taskbar.

- To add a task, click the toolbar New button. A new task field appears in the task list; type a name for the task in this field.
- To delete a task, select the task name in the list, and then click the Delete button on the toolbar.
- To arrange the order of the icons in your taskbar, select the task name in the list, and then click the toolbar Move Up or Move Down buttons.

Script to Run—You can locate and specify a program to run for your task, by clicking the button next to the Script to Run field to open the Windows Explorer file chooser.

Clicking the Change Icon button opens the Change Icon dialog. You can choose an icon to display in the taskbar for the task.



Package Distribution Report

From the taskbar on the workbench, you can generate a report showing the location of packages in a project by clicking once the Package Distribution Report icon in the taskbar. A Microsoft Excel report shows in the list view. A functional Excel toolbar is added to the workbench and removed when you close the report. The menu bar provides options to save, print and so on.

Note: Set your context before generating this report.

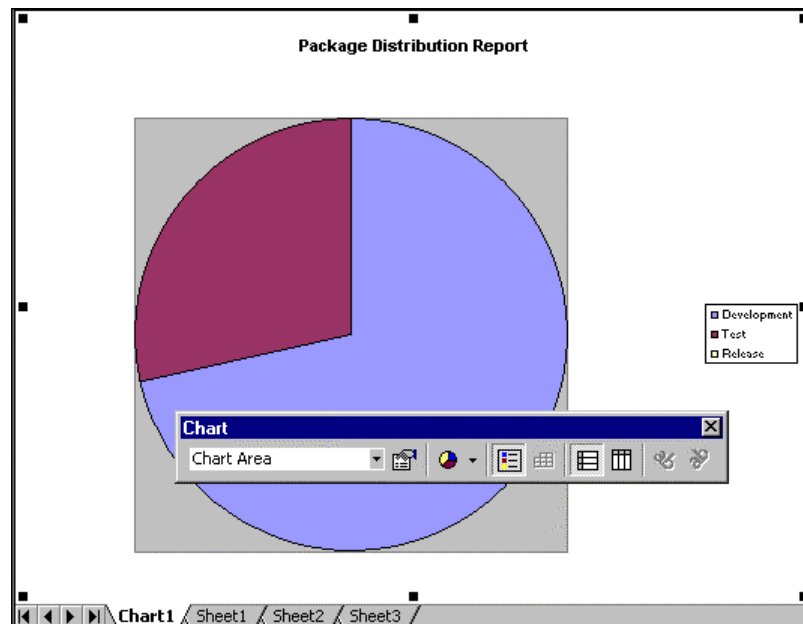


Chart Toolbar

A variety of options to customize the report are available from the toolbar or shortcut menu. You can depict the report in eighteen different chart types by choosing an option from the Chart Type drop-down list. You can format the chart in a variety of styles and choose the objects you want included. Other options allow you to show a legend, show data in columns or rows, and angle text.

Administrator Reports

From the taskbar on the Administrator window, you can generate reports showing access for projects or repositories-Project Access Report, Repository Access Report, respectively. After clicking once the shortcut icon in the taskbar, a dialog prompts for the project or repository name you want to report on. Enter a name (the asterisk wildcard is allowed) and click the OK button to generate the report. A Microsoft Word report shows in the list view. A functional Word toolbar is added to the workbench and removed when you close the report. The menu bar provides options to save, print and so on. See the chapter "Object Access Control Administration" for examples of these reports.

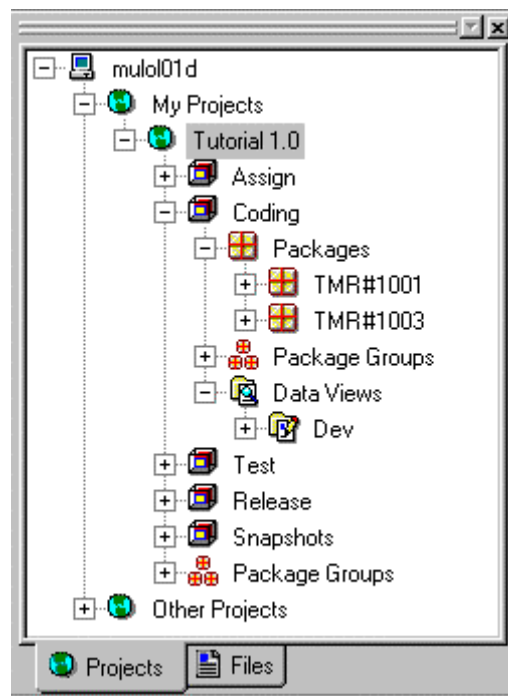
Workspace

The workspace is the middle pane on the main window. Depending on whether you are logged in to the workbench or to the Administrator window, the workspace contains different tabs.

- On the Administrator Window, the workspace shows five tabs: Life cycles, Repositories, User Groups, Forms and Files. The Administrator window provides you with the functions you need to set up life cycles and perform administrative activities.
- On the workbench, the workspace shows two tabs: Projects and Files. The workbench provides you with the functions you need to perform day-to-day activities.

Projects Tab

Using the Projects tab, you can perform most of your day-to-day Harvest activity, such as creating a package and form, checking items in and out, and moving packages through the life cycle.



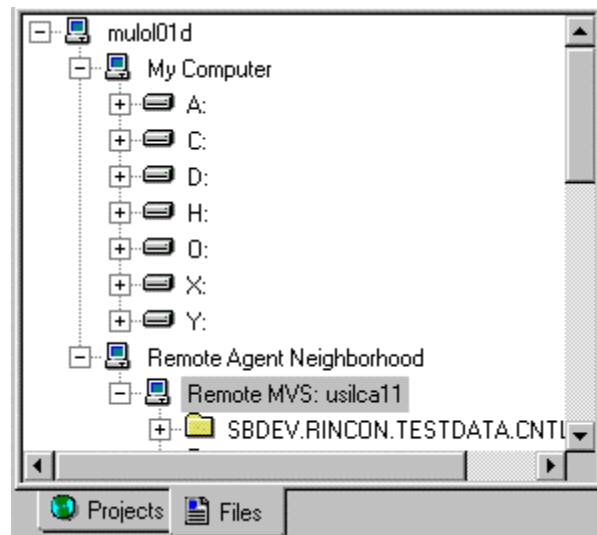
On this tab, two folders are used to access and organize your projects:

- *My Projects* can be used to easily locate the projects on which you customarily work.
- *Other Projects* contains all projects, both active and inactive.

You can easily add and remove projects from My Projects and Other Projects, either by dragging a project or by right-clicking a project and choosing the move option on the shortcut menu. For information about this function, see the *Procedures Guide*.

The Project Lifecycle Viewer gives you a graphical view of the development life cycle (Harvest project). Project states are depicted as boxes with a colored bar across the top that depicts the View type. Harvest promote and demote processes are represented as connecting, directed lines. You can open the Project Lifecycle Viewer by right-clicking the Projects folder and choosing View Project Lifecycle from the shortcut menu. For more information about the Project Lifecycle Viewer, see the *Administrator Guide*.

Files Tab



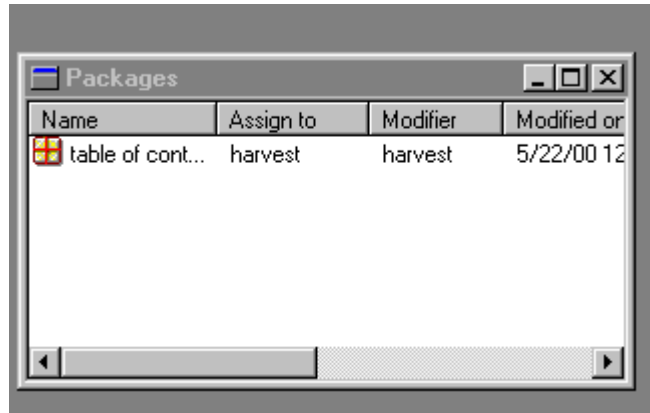
The Files tab displays all of the files that you can access on your client workstation.

- The *My Computer* folder shows the drives that are mapped to your local machine.
- The *Remote Agent Neighborhood* folder shows all available file agents. Using the Files tab, you can check in files from remote locations. See the chapter "The Harvest Agent" for information about using the agents.

To easily locate an agent in the workspace, you can place it in your neighborhood. The agents remain in your neighborhood until you remove them.

List View

When an object is selected in the workspace, the properties or contents of that object are displayed in the list view.



The list view operates in the following ways:

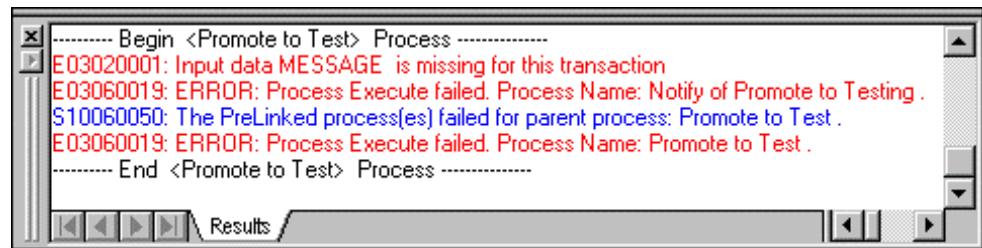
- Each time you click an object or folder in the workspace, the active list view window is refreshed with the contents of the new selection. For example, double clicking the Package Groups folder in the workspace refreshes the contents of the active window and shows the package groups belonging to the project or state.
- When you double click a package in the list view, the folder to which it belongs is automatically opened in the workspace.

From the versions list on the workbench, you can view a version's content by right-clicking a version and choosing View Version from the shortcut menu. The editor of the file's origination displays the file in read-only mode in the list view; for example, a Microsoft Word file would use Microsoft Word.

Output Log

Most activities in Harvest generate output that is sent to the output log. This output can be the focus of the activity, such as in generating reports. At other times, it is informational and documents the results or consequences of a previous action. Error messages are shown in red text for high visibility.

Whenever new information is sent to the output log, it is appended to the end. Its contents are maintained throughout the session, unless it is cleared or closed. Information in the output log can be cleared, copied, printed, or saved.



A right-click on the output log opens a shortcut menu that provides these functions: Allow Docking, Hide, Save As, Print, Clear, Copy and Select All.

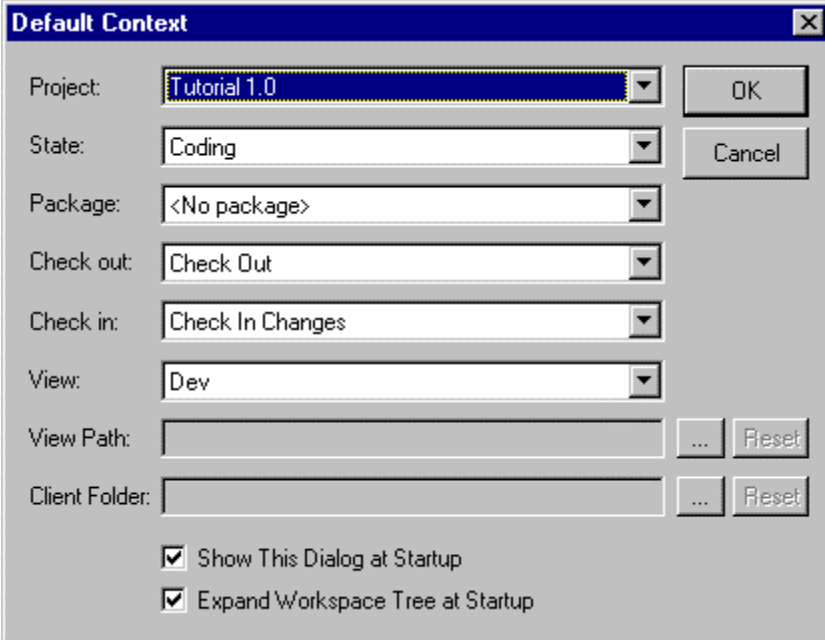
- Allow Docking allows you to detach the log and place it elsewhere. After choosing Allow Docking, to undock the output log, double click it or place your cursor on the output log and drag it to a different location. To dock the output log, drag it back to the main window or double click it.
- Hide hides the output log. To view the output log again, choose View, Output from the menu bar.
- Save As opens the Save As window that enables you to name the output file and save it in a specified location.
- Print opens the system Print dialog.
- Clear removes all information from the output log.
- Copy allows you to copy the output information and paste it in another location, such as in a Microsoft Word file.
- Select All selects all information in the output log.

Context

Context is your project, state, package, or snapshot selection within Harvest. Your current context is your current location and determines what is displayed and available according to your access and privileges. For example, selecting a state enables you to view packages, package groups, versions, and forms. Each process available within the selected state appears under the Processes menu and you can invoke any process for which you have access.

The Default Context dialog enables you to set a default context that automatically opens the workspace tree on the workbench to a specific context. You can also set a default context to automatically synchronize the check in and check out processes with the client directory and view path structure. Using the Default Context dialog, you can confirm or modify your settings.

By default, the Default Context dialog opens after logging in to the workbench. If you prefer logging directly into the workbench, you can thereafter bypass the dialog by disabling the Show This Dialog at Startup button. The Default Context dialog can also be invoked by choosing Tools, Set Context, clicking the Set Context toolbar button, or by right-clicking a project or state and choosing Set Default Context from the shortcut menu.

The image shows a Windows-style dialog box titled "Default Context" with a close button (X) in the top right corner. The dialog contains several fields and checkboxes. The fields are: "Project:" with a dropdown menu showing "Tutorial 1.0"; "State:" with a dropdown menu showing "Coding"; "Package:" with a dropdown menu showing "<No package>"; "Check out:" with a dropdown menu showing "Check Out"; "Check in:" with a dropdown menu showing "Check In Changes"; "View:" with a dropdown menu showing "Dev"; "View Path:" with a text box and a browse button (...); and "Client Folder:" with a text box and a browse button (...). To the right of the "Project:" and "State:" fields are "OK" and "Cancel" buttons respectively. Below the "View Path:" and "Client Folder:" fields are "Reset" buttons. At the bottom of the dialog are two checked checkboxes: "Show This Dialog at Startup" and "Expand Workspace Tree at Startup".

Project—The Project field is automatically populated with your current project location. To change the project, use the drop-down list to choose a destination project.

State—The State field is automatically populated with your current state location. To change the state, use the drop-down list to choose a destination state.

Package—The Package field is automatically populated with your current package selection. To change the package, use the drop-down list to choose a package.

Check out—Choose a default check out process from the drop-down list. The list shows all check out processes defined for your current state.

Check in—Choose a default check in process from the drop-down list. The list shows all check in processes defined for your current state.

View—The View field is automatically populated with your current view. To change the view, use the drop-down list to choose a working or snapshot view. The view selected in this field determines the paths available in the View Path field.

View Path and Client Folder—The View Path and Client Folder fields work together to help you synchronize the internal and external file structures. They are typically used to establish a point at which paths in the repository mirror working directories on the client.

Synchronizing the client directory and view path can be used with recursive check out and check in. During a recursive operation, Harvest starts at a specified client directory or view path and searches all directories or paths below it for files or items to operate on. For more information about recursive operations, see the chapter "Understanding Change Management."

The way you set your default context is flexible. You can set either or both fields. The settings remain in effect until you establish a context on the workbench by selecting a file or version and execute a check in or check out process. Your current context is then used in the execution dialog, overriding the Default Context dialog settings.

View Path—The view path specifies a location in the repository from which items are checked out or files are checked in. By specifying a view path, you are setting a default path that is placed in this field in subsequent check outs and check ins. To change your repository path, click the button next to the View Path field. This invokes the Select a Repository Item Path dialog that allows you to browse through available paths and select a location.

Client Folder—The directory selected for the Client Folder field specifies the destination on the file system from which files are checked in or to which items are checked-out. If either of the preserve structure check out options is being used, the check out specifies a path and this specification appends to the file system directory. To change the destination, click the button next to the Client Folder field. This invokes the Select a Directory Path dialog that allows you to navigate through directories available on your system to select a file.

Show This Dialog at Startup—By default, this option is enabled. After logging in to the workbench, the Default Context dialog opens. If you prefer logging directly into the workbench, you can thereafter bypass the dialog by disabling this option.

Expand Workspace Tree at Startup—By default, this option is enabled. This option causes the workspace tree on the workbench to be opened to the context you specify in this dialog, after you log in.

Kinds of Dialogs

Most dialogs in Harvest fall into one of three general categories: Find and Select, Process Execution, and Properties. These kinds of dialogs are described below.

- Find and Select

Select dialogs allow you to select an appropriate value for a field on another dialog. Select dialogs simply list the available objects of a type.

Find dialogs: Find Form, Find Package, and Find Version provide multiple filtering options for delimiting the listed objects, and can be invoked from the Tools menu and the toolbar.

- **Process Execution**

Process Execution dialogs allow you to execute a process. The name displayed in the title bar of a Process Execution dialog is the same as the name defined by the Harvest administrator on the Properties dialog for the process.

- **Properties**

Each object in Harvest has an associated properties dialog. Properties dialogs are used for defining, modifying or viewing the attributes of objects.

Single-User Agent

The Harvest agent enables you to access remote file systems from the Harvest client. For example, you can execute the check out processes to a remote UNIX filesystem from a Windows Harvest client. This chapter provides information on:

- Starting a single-user agent.
- Connecting to an agent.
- Using an agent.

The Harvest agent allows users to login to a remote filesystem and perform check in and check out functions. The agent can be started in either of two modes: as a multi-user or single-user agent process. A privileged user account is required to start the agent in multi-user mode, that is, “root” on UNIX systems and “Administrator” on Windows systems. The single-user agent does not require a privileged user account for startup. The user who started it owns the single user agent process. Therefore, only the user who owns the single user agent process can login. See the sections in this chapter for information on starting a single-user agent. See the *Administrator Guide* for information on the multi-user agent.

Starting a Single-User Agent

You can start a single-user agent by selecting Agent from the Harvest program group, or from the command line type: **agntd -option**

You can specify the agent start options either on the command-line or in the agent argument file, HAgent.arg, located in the HARVESTHOME directory. Options specified on the command-line will override the same options set in the hagent.arg file.

The Harvest single-user agent supports the following start options:

Option	Description
-usr=<login name>	Host system login name. The username specified must be the name of the user currently logged into the host machine.

Option	Description
-pwd=<password>	A user defined password. This password does not have to be the host system user password. It can be a password defined per agent process instance. If this option is not specified the user will be prompted for a password. See also -pwdmethod.
-timeout=<integer> Example: -timeout=10	This option specifies the agent idle time limit in minutes. When the idle timeout limit is reached, the agent process will shut down automatically. If -timeout=none is specified or the -timeout option is not specified, the agent process will not time out.
-pwdmethod=<string> Example: -pwdmethod=prompt -pwdmethod=random	The password method option takes two values: prompt or random. The -pwd option overrides this option. The prompt string value will prompt the user for the password. The random string value will generate a random password string and display it on the screen.
-rtserver=<machine name>	Connect to PEC network on the machine. The machine name is typically the hostname of the Harvest broker machine.
-homedir=<harvest home dir>	Specify location for argument/log file
-verbose	Display the log information on the screen.
-help	Display the options description.

The agent thread options, -minthread, -maxthread and -tmploop, are disabled for single-user agent mode. The single-user agent utilizes one process thread.

Starting a Single-User Agent on Windows Systems

Single-User Agent Startup Example

To start a single-user agent, the -usr option must be specified either on the command-line or in the hagent.arg file. The username specified to the -usr option **must** be the name of current user logged into the host machine. If the option -pwd is not specified you will be prompted for a password at startup. For example:

```
C:\>agntd -usr=harvest -rtserver=sparc
PASSWORD:****
Creating windowless child process ...
Parent process exiting ...
C:\>
```

where the password specified is not the password for the host system account “harvest”, but a session password defined at runtime for this agent process. A different session password can be defined each time a single-user agent process is started. The `-rtserver` option is set to the hostname of the Harvest broker machine.

Note: On Windows 2000/XP systems, the username is case-sensitive. On Windows NT systems, the username is not case-sensitive.

Note: If the username specified to the `-usr` option is not the same as the current user logged into the host machine, you will receive an “incorrect user name” startup error. For example:

```
HAgent | 20020605 07:59:24 | ERROR: Incorrect user name
```

Each user can start many single-user agent processes on one machine and connect to different Harvest brokers (rtservers), for example:

```
agntd -usr=harvest -rtserver=sparc
agntd -usr=harvest -rtserver=alpha
agntd -usr=harvest -rtserver=bigblue
...
```

Only one user can start a single-user agent process on the same machine and connect to a single Harvest broker (rtserver), for example:

```
agntd -usr=harvest01 -rtserver=sparc
agntd -usr=harvest02 -rtserver=sparc
agntd -usr=harvest03 -rtserver=sparc
...
```

Specifying the Single-User Agent Password

The single-user agent password can be specified either by the user starting the agent process or it can be automatically generated.

The password can be specified by the user starting the agent with or without using the `-pwd` option:

- The `-pwd` option can be specified in the `hagent.arg` file or on the command-line to explicitly set the single-user agent session password.
- If the `-pwd` option is not specified, the user will be prompted for a session password (specifying `-pwdmethod=prompt` is the same as not specifying `-pwd`).

The single-user agent password can be automatically generated using the `-pwdmethod=random` option. This option will generate a random session password and display it to the standard output device (terminal by default). For example:

```
C:\>agntd -usr=harvest -pwdmethod=random
Harvest Agent Generated Password: 4zQBcNmX
Creating windowless child process ...
Parent process exiting ...
C:\>
```

Single-User Agent Log Files

A log file is generated on startup in the `%HARVESTHOME%\log` directory. The naming format is:

```
YyyymmddHAgent_UserName_RTserverName.log
```

where `RTserverName` is the value of the `-rtserver` option or `yyyymmddHAgent_UserName.log`

if no `-rtserver` option is specified on the command-line or in the `hagent.arg` file.

Starting a Single-User Agent on Unix Systems

Single-User Agent Startup Example

To start a single-user agent, the `-usr` option must be specified either on the command-line or in the `hagent.arg` file. The username specified to the `-usr` option **must** be the name of current user logged into the host machine. If the option `-pwd` is not specified you will be prompted for a password at startup. For example:

```
$/agntd -usr=harvest -rtserver=sparc
PASSWORD:****
Creating daemon child process ...
Parent process exiting ...
$
```

where the password specified is not the password for the host system account “harvest”, but a session password defined at runtime for this agent process. A different session password can be defined each time a single-user agent process is started. The `-rtserver` option is set to the hostname of the Harvest broker machine.

Note: The user account starting a single-user agent process must have write access to the `$HARVESTHOME/log` directory. If write access is not granted, a log file write error will occur. For example:

```
Error : can not open log file <HAgent_user> in <HARVESTHOME>
```


Note: If the username specified to the `-usr` option is not the same as the current user logged into the host machine, you will receive an “incorrect user name” startup error. For example:

```
HAgent | 20020605 07:59:24 | ERROR: Incorrect user name
```

Each user can start many single-user agent processes on one machine and connect to different Harvest brokers (rtserver), for example:

```
agntd -usr=harvest -rtserver=sparc
agntd -usr=harvest -rtserver=alpha
agntd -usr=harvest -rtserver=bigblue
...
```

Only one user can start a single-user agent process on the same machine and connect to a single Harvest broker (rtserver), for example:

```
agntd -usr=harvest01 -rtserver=sparc
agntd -usr=harvest02 -rtserver=sparc
agntd -usr=harvest03 -rtserver=sparc
...
```

Specifying the Single-User Agent Password

The single-user agent password can be specified either by the user starting the agent process or it can be automatically generated.

The password can be specified by the user starting the agent with or without using the `-pwd` option:

- The `-pwd` option can be specified in the `HAgent.arg` file or on the command-line to explicitly set the single-user agent session password.
- If the `-pwd` option is not specified, the user will be prompted for a session password (specifying `-pwdmethod=prompt` is the same as not specifying `-pwd`).

The single-user agent password can be automatically generated using the `-pwdmethod=random` option. This option will generate a random session password and display it to the standard output device (terminal by default). For example:

```
$/agntd -usr=harvest -pwdmethod=random
Harvest Agent Generated Password: 4zQBcNmX
Creating daemon child process ...
Parent process exiting ...
$
```

Single-User Agent Log Files

A log file is generated on startup in the \$HARVESTHOME/log directory. The naming format is:

YyyymmddHAgent_UserName_RTserverName.log

where RTserverName is the value of the -rtserver option or

yyyymmddHAgent_UserName.log

if no -rtserver option is specified on the command-line or in the hagent.arg file.

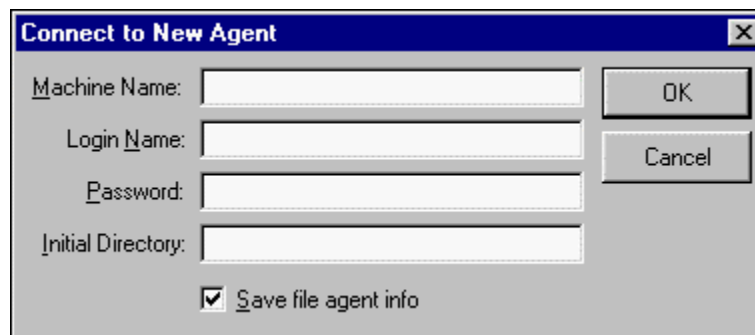
Connecting to an Agent

To access a remote agent, you must first establish a connection to that agent. After connecting to an agent, the remote machine's directories and files are displayed and accessible on the Files tab of the workspace.

The agent login function will look up the available agents on the specified machine and try to connect to the Single-User Agent first. If the Single-User agent does not exist or login fails then a Multi-User agent login is attempted.

To connect to a remote agent:

1. On the Files tab of the workbench, click the plus sign (+) beside the Remote Agent Neighborhood to open it.
2. Right-click on the "Remote Agent Neighborhood" and choose Connect to New Agent from the shortcut menu. A login window opens.



3. Enter the name of the machine you want to log in to.
4. Enter the login name and password for the remote agent.
5. Enter the full path to the initial directory.

For UNIX systems, if the initial directory is not specified you will be logged into the user home directory.

For MVS remote file systems, you must specify the MVS prefix in the Initial Directory field of the Connect to New Agent dialog.

Note: Once an agent is created you can change the initial directory by right clicking the agent and choosing Change initial directory from the shortcut menu.

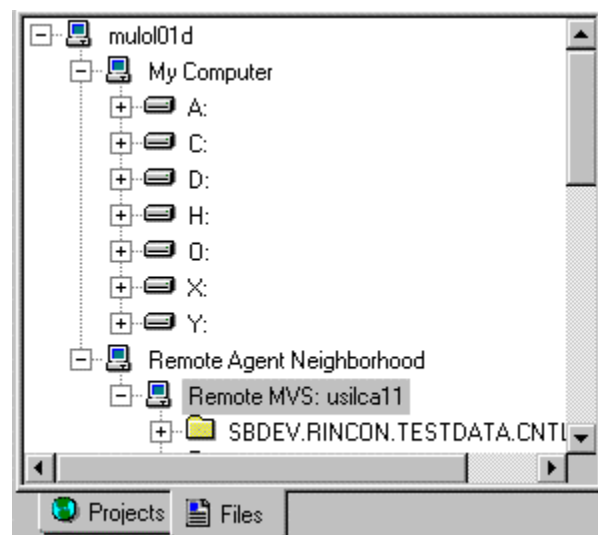
6. Save file agent info: click to clear this check box if you do not want to connect to the remote file agent the next time the Harvest Workbench is started.

Using the Agent

Workbench

The Files tab displays all of the files that you can access on your client workstation.

- The *My Computer* folder shows the drives that are mapped to your local machine.
- The *Remote Agent Neighborhood* folder shows all available file agents.



Using the Files tab, you can check in files from remote locations. The check in process allows you to check in files from the client directory system into Harvest, creating new items or updating existing items that have changed. For example, you can check in the files on an NT or MVS platform into a Harvest repository that is managed by a server process that runs on a Solaris platform where the file cannot be accessed through NFS. Access to data files in remote locations requires a user name and password for the remote machine.

Remote Agent Connections From the Command-Line

The Harvest command-line check out (hco) supports connecting to a remote agent. For example, the hco command can be used with a remote agent connection to perform a check out to a remote file system. The following example shows one possible syntax for a remote agent check out. The hco command is run from a Windows client to check out to a UNIX filesystem:

```
hco -b "win01" -en "Production" -st "Dev" -vp "\ProdRep"  
-cp "/users/harvest" -br -op as -s "*" -usr "harvest" -pw "harvest"  
-rm "sparc" -rusr "harvest" -rpw "test" -o "c:\harvest\log\hco.log"
```

See the *Reference Guide* for detailed descriptions on the Harvest command-line options.

Agent Shutdown

The Harvest agent shutdown can be controlled by the Harvest broker shutdown. The broker shutdown option, -shutdown=all, will shutdown all Harvest agent processes that are visible in the RTserver network.

The Single-User Agent can be shutdown using an idle timeout option specified at agent startup, -timeout=<integer>. Where <integer> is the agent idle time limit in minutes. When the idle timeout limit is reached, the agent process will shut down automatically.

Packages and Forms

This chapter explains how *packages*, *package groups*, and *forms* enable you to maintain and organize information and changes within Harvest.

Packages, package groups, and forms are accessed using the Projects tab on the workbench.

Packages

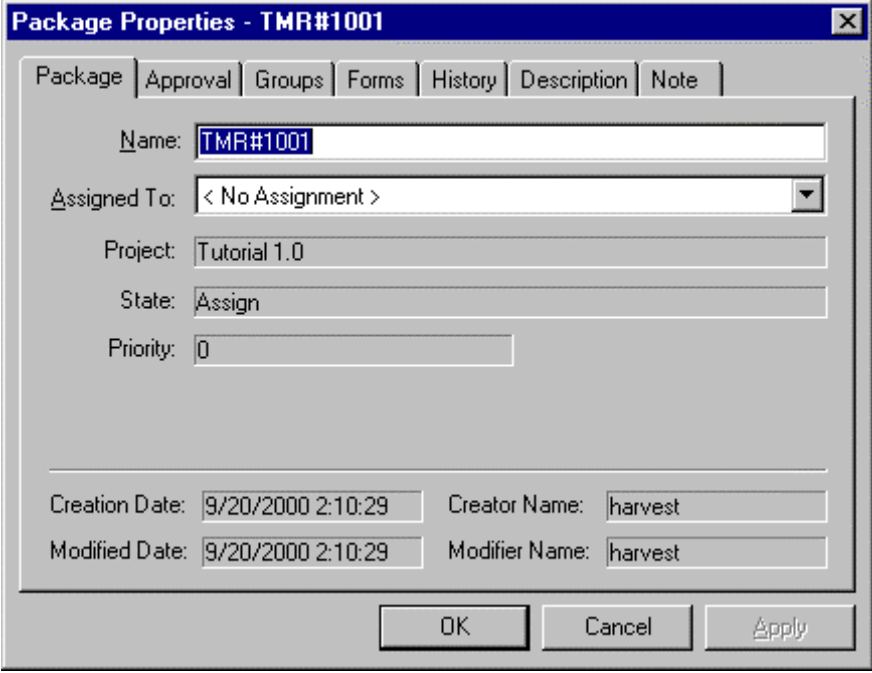
In Harvest, all version changes must be made in reference to a package. A package is the basic unit of work that moves through the life cycle. It typically represents a problem, task or an incident that needs to be tracked, the changes made in response to the problem, task or incident, and any associated information.

Some of the important aspects of a package include:

- **Package Group.** A package can be added to one or more package groups. Almost any action that can be performed on a package can be performed on a package group.
- **State.** Once defined, the state of a package can be changed only through promote and demote processes.
- **Forms.** One or more forms can be associated with an existing package. The form provides the link between Harvest's change management functions and problem tracking.
- **Versions.** One or more versions can be associated with a package. Double-clicking a package displays the versions associated with the package.

Package Properties Dialog

The Package Properties dialog is used to define a package and to associate the package to one or more package groups. The Package Properties dialog is shown below.



The screenshot shows the 'Package Properties - TMR#1001' dialog box. It has a tabbed interface with tabs for 'Package', 'Approval', 'Groups', 'Forms', 'History', 'Description', and 'Note'. The 'Package' tab is selected. The dialog contains the following fields:

- Name:** TMR#1001
- Assigned To:** < No Assignment > (dropdown menu)
- Project:** Tutorial 1.0
- State:** Assign
- Priority:** 0
- Creation Date:** 9/20/2000 2:10:29
- Creator Name:** harvest
- Modified Date:** 9/20/2000 2:10:29
- Modifier Name:** harvest

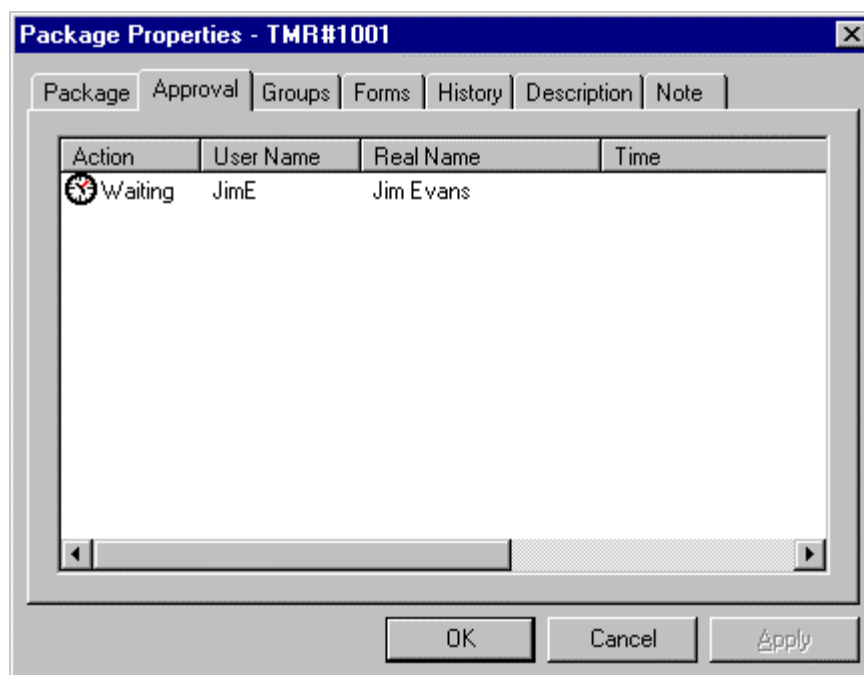
At the bottom right, there are three buttons: 'OK', 'Cancel', and 'Apply'.

Name—The name of the package is displayed in the Name field and can be edited.

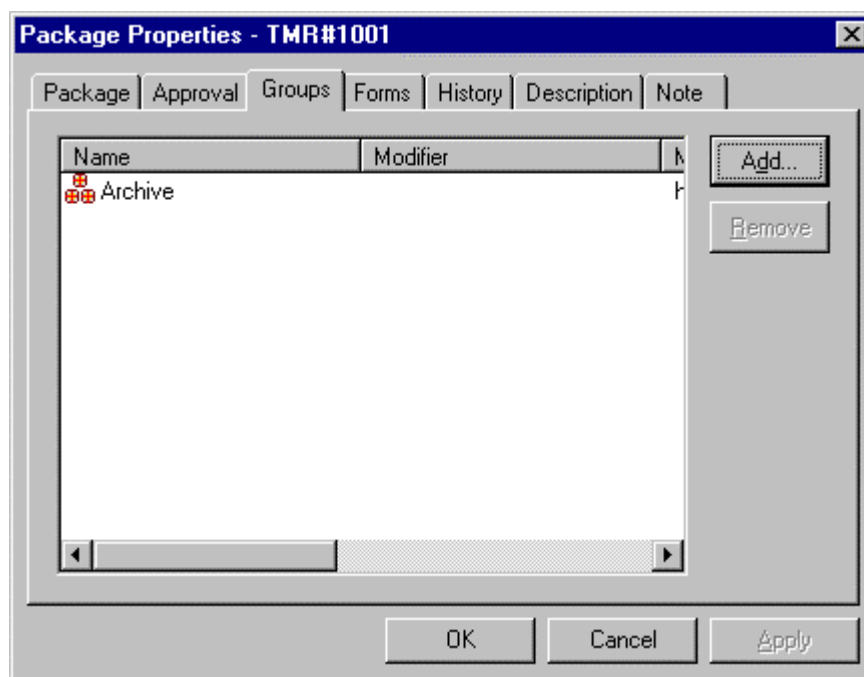
Assigned To—This field displays the package assignee. You can change the assignment by using the drop-down list to select a user from a list of all defined Harvest users.

Project and State—The project and state to which the package belongs are also displayed.

The Approval tab lists the actions taken on a package and its current status.



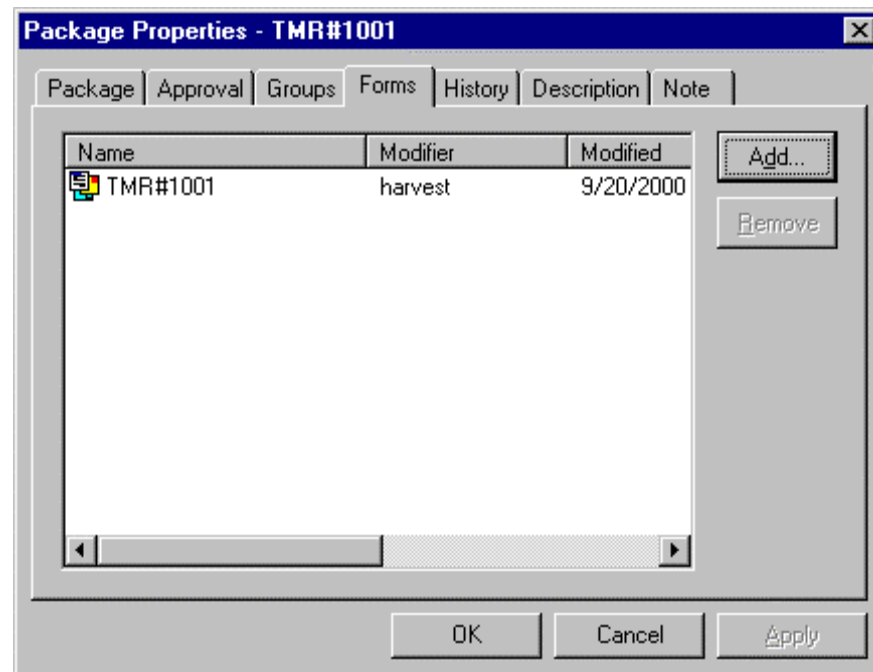
The Groups tab lists the package groups associated with the current package and allows you to create or modify associations.



Clicking the Add button opens a dialog that lists all the package groups in your current project or state. You can select a package group to add to the list of package groups associated with the current package.

To remove a package group from the list of package groups associated with the current package, you can select the package group, and then click the Remove button. This does not delete the selected package group; it simply removes the association.

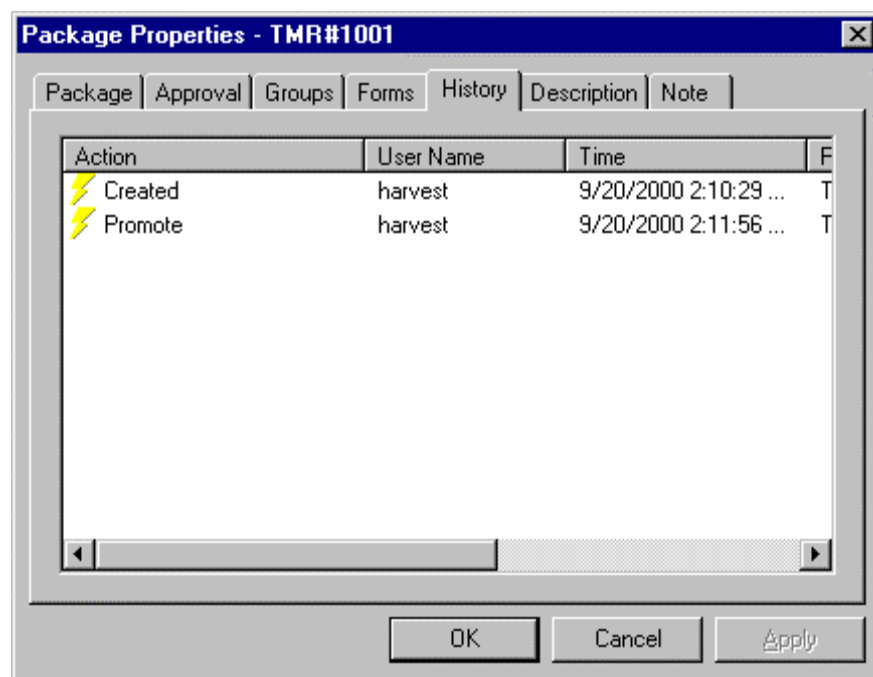
The Forms tab lists the forms associated with the current package and allows you to create or modify associations.



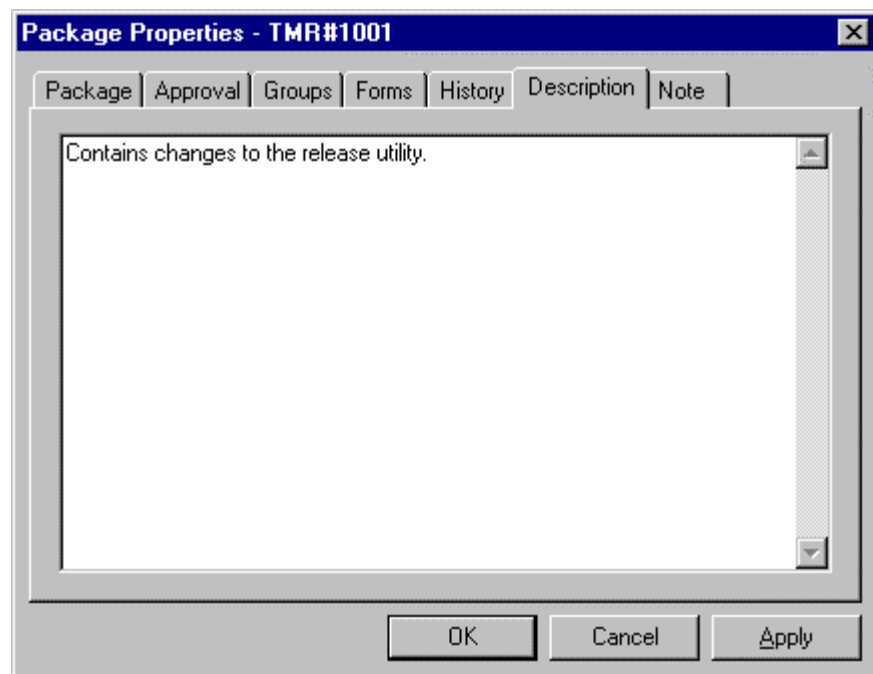
Clicking the Add button opens the Find Form dialog. Using the Find Form dialog, you can locate and select a form to add to the list of forms associated with the current package.

To remove a form from the list of forms associated with the current package, you can select the form, and then click the Remove button. This does not delete the selected form; it simply removes the association.

The History tab shows the history of the current package.



On the Description tab you can enter a description of the package.



Package Groups

A *package group* consists of two or more packages. Package groups can be used to organize types of packages. For example, all the packages that affect the user interface of an application could be put in a group named GUI_Project. If some of these packages require documentation modifications as well, they could also be put in the Documentation package group because a package can be in multiple groups.

Almost all operations that can be performed on a package can also be performed on a package group. For example, a user can approve and promote an entire package group.

Package groups are flexible. They can be created and deleted, and packages can be added or removed from them at any time. This allows packages to be grouped dynamically. This type of grouping can be used to define sub-projects, which can then be manipulated as a unit. For example, assume that all packages in a group called Phase_1 are in the QA state. If a problem is found in one of the packages in that group, that package can be demoted to a previous state where more work can be done on it, and then promoted back to the QA state to join the group. In this way, a sub-project can be managed as a group, yet smaller parts can also be managed individually.

The *bind packages* feature enforces the following restrictions to packages belonging to a package group:

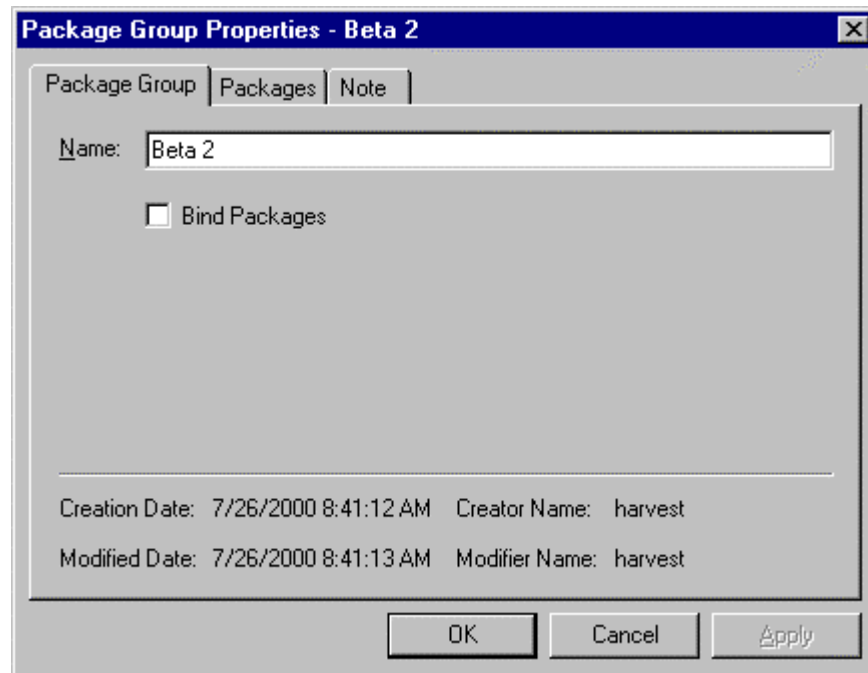
- A package can belong to only one bound package group.
- If a package is part of a bound package group and if Enforce Bind is enabled on the Demote or Promote Properties dialog, all packages in the group must be demoted or promoted together.

You cannot promote or demote a single package that belongs to a bound package group while Enforce Package Bind is enabled. To promote or demote a single package that is a member of a bound package group, you must either not check Enforce Package Bind or remove it from the bound package group.

Package groups can also be used to generate reports. Reports can be generated on the status of packages in a specified group.

Package Group Properties Dialog

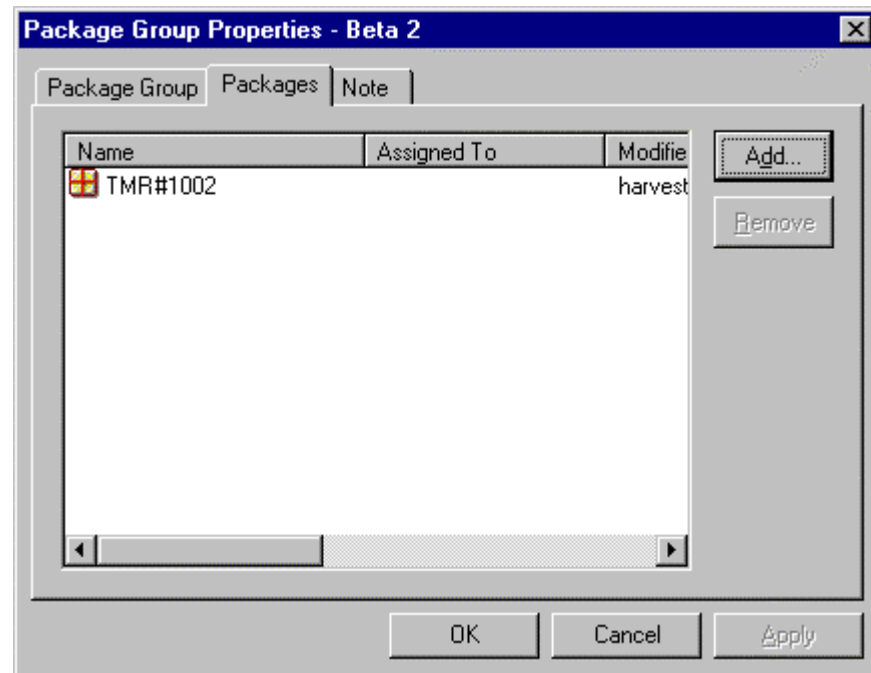
The Package Group Properties dialog is used to define a package group and to associate the package group to one or more packages.



Name—The name of the package group is displayed in the Name field and can be edited.

Bind Packages—To enforce the Bind Packages restrictions, enable the Bind Packages check box.

The Packages tab lists the packages associated with the current package group and allows you to create or modify associations.



Clicking the Add button opens a dialog that lists all the packages in your current project or state. It allows you to select a package to add to the list of packages associated with the current package group.

To remove a package from the list of packages associated with the current package, select the package, and then click the Remove button. This does not delete the selected package; it simply removes the association.

Forms

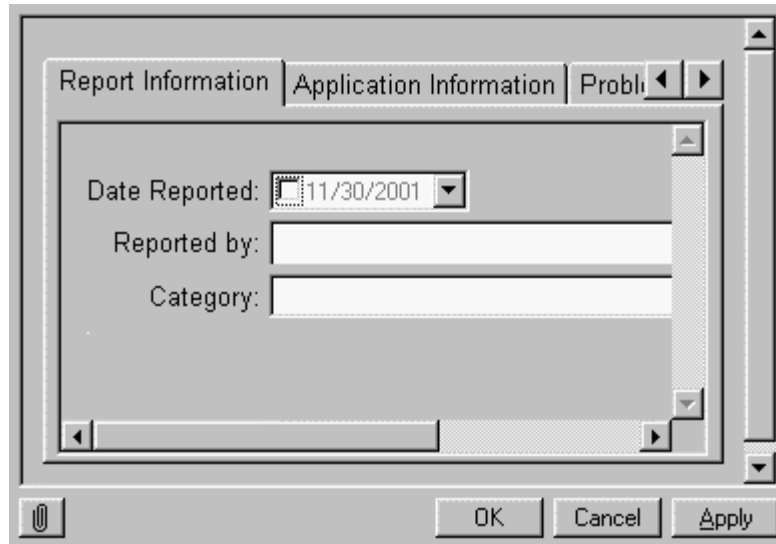
Harvest forms are used in a similar way as paper forms. For example, a form can be used to track support issues or software failures. A form could also help you organize information for a project or about a specific customer. The form's modification history is maintained and can be easily viewed. You can create form attachments to provide additional relevant information, see the section "Form Attachments" in this chapter. Every form must have an associated package, see the section "Package and Form Associations" in this chapter for details.

Forms are displayed in the Form Viewer when you double-click on a form or use the Find Form dialog to locate a form. The fields displayed vary according to the type of form loaded.

When displaying a form in the Form Viewer you can print the form in a list format by clicking on the print toolbar icon or choosing File, Print.

Forms can be edited concurrently. If a form has been modified while another user was editing the form, the differences are displayed in a side-by-side list. The user must then choose the form values to save.

The Form Viewer containing a form is shown below.

The image shows a screenshot of a 'Form Viewer' dialog box. At the top, there are three tabs: 'Report Information' (which is selected), 'Application Information', and 'Problem'. Below the tabs, there are three input fields: 'Date Reported:' with a date picker showing '11/30/2001', 'Reported by:' with an empty text box, and 'Category:' with an empty text box. At the bottom of the dialog, there is a toolbar with a printer icon, and three buttons: 'OK', 'Cancel', and 'Apply'.

You can toggle between the list view and form viewer by using the keys Ctrl-Tab. For example, you could have a list of packages displayed in the list view, a maximized form overlaying it, and be able to switch between the form and package list by typing <Ctrl-Tab>.

Default Form Types

Harvest supplies the following default form types:

- **Application Change Request** is used to record the change process within third party development projects.
- **Comment** is used to record any kind of information and associate it with a package or other form.
- **Defect Tracking** is used to track defects in an application. It contains many fields useful in a help desk situation.
- **ESD Change Request** is used to manage the changes made to Server and Desktop file packages.
- **Modification Request** is used for any type of software change request.
- **Problem Report** is used for organizing and tracking information about a software problem.
- **Q and A** is used to create a database of information in question and answer format. This can be an effective support tool for a help desk or customer support group. Keyword searches can be performed to retrieve information.
- **Testing Info** is used to support the testing of packages. Developers can record information useful for users who perform tests and quality assurance. Testing and quality assurance personnel can record their responses for management review or to assist developers if the package is demoted.
- **User Contact Form** is used to organize and track information about your customers or product users.

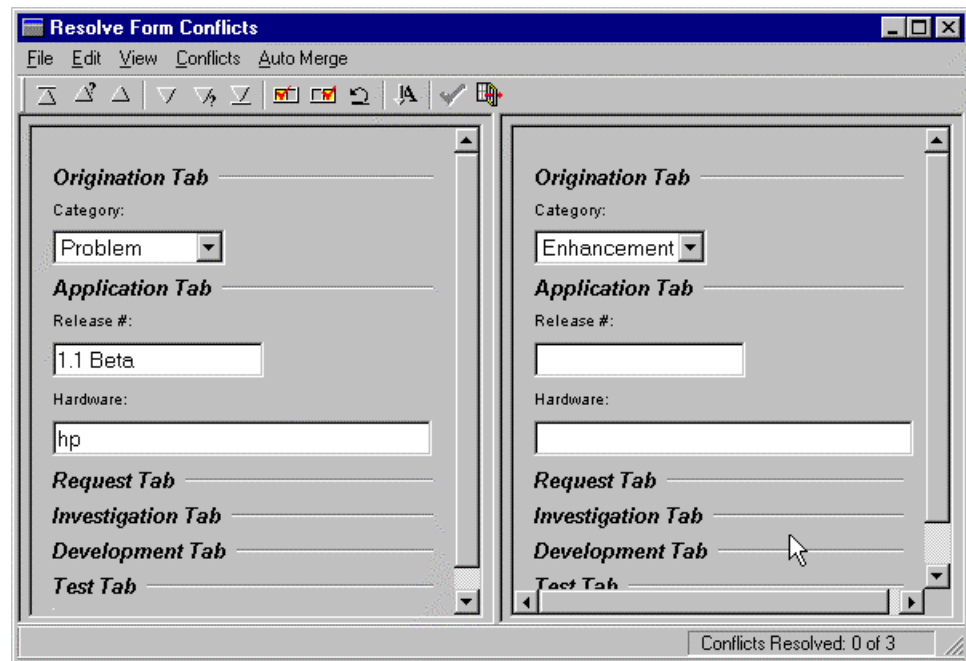
For detailed information about each form type, see the *Administrator Guide*.

In addition to these form types, custom form types can be created and added to Harvest by using the Form Wizard. See the *Administrator Guide* for details.

Concurrent Update

Two users can update a form at the same time. The user who saves the form last (current form) is presented with the Resolve Form Conflicts dialog if differences exist between the two forms. The Resolve Form Conflicts dialog enables the current user to view and merge the differences between the two forms.

The two versions of the form being merged are displayed in two panes. The panes are positioned side-by-side and show only fields that differ between the forms. Your current form is shown in the left pane and the previously saved version of the form is shown in the right pane. The panes are synchronized to ensure that the two panes are displaying the same fields. One field must be selected in favor of the other.



Field outline color codes indicate the status of the form merge.

- Blue indicates your current field location.
- Red indicates resolved fields.

Status Bar—The status bar at the bottom of the dialog contains the Resolved meter. This meter shows the number of resolved conflicts and the total number of conflicts.

Navigation

The Conflicts menu and a toolbar provide you with different options to navigate through the conflicts. The navigation options found in the Conflicts menu are the same as those found on the toolbar. The toolbar can be undocked and relocated on the dialog.

You have the following options:

- **First** moves you to the first conflict.
- **Previous Unresolved** moves you to the previous unresolved conflict.
- **Previous** moves you to the previous conflict.
- **Next** moves you to the next conflict.
- **Next Unresolved** moves you to the next unresolved conflict.
- **Last** moves you to the last conflict.
- **Auto Jump** automatically moves you to the next field after you resolve a conflict.

View

The View menu lets you customize the dialog to show or hide the Merge Tools, Toolbar, and Status Bar.

Resolving Conflicts

Options to select the right or left pane fields are available from the Edit menu, shortcut menu, shortcut keys and the toolbar.

- **Take Left** chooses the conflict on the left pane (your current form).
- **Take Right** chooses the conflict on the right pane (the previously saved version of the form).

Two options allow you to reset conflicts to an unresolved condition.

- **Reset Current Conflict** restores the current conflict field to its original condition. This option is available from the toolbar and the Edit menu.
- **Reset All Conflicts** restores the all conflict fields to their original condition. This option is available from the Edit menu.

Cut <Ctrl+X>, copy <Ctrl+C>, and paste <Ctrl+V> functions are active in the Resolve Form Conflicts dialog and can be used in any text field. All fields are editable and any changes can be made to resolve conflicts.

Auto Merge Options—From the drop-down list, you can select **one** of the Auto Merge options to specify how the process executes.

- **Take All Changes from Left Pane** automatically selects your current form fields to create the final version.
- **Take All Changes from Right Pane** automatically selects the previously saved version of the form to create the final version.

To reset the Resolve Form Conflicts dialog to its original contents, click the Cancel button.

You cannot close the Resolve Form Conflicts dialog until all conflicts are resolved; the Save option is disabled until all conflicts are resolved. A green check mark indicates that all conflicts are resolved and you are ready to save the final edits. When all conflicts are resolved, select Save to save the form; this replaces the previous version with the new version.

Form Attachments

Each form can have one or more attachments associated with it, linking the form to additional relevant information. You can create form attachments, and by selecting an attachment you can reference web sites, view files or copy files.

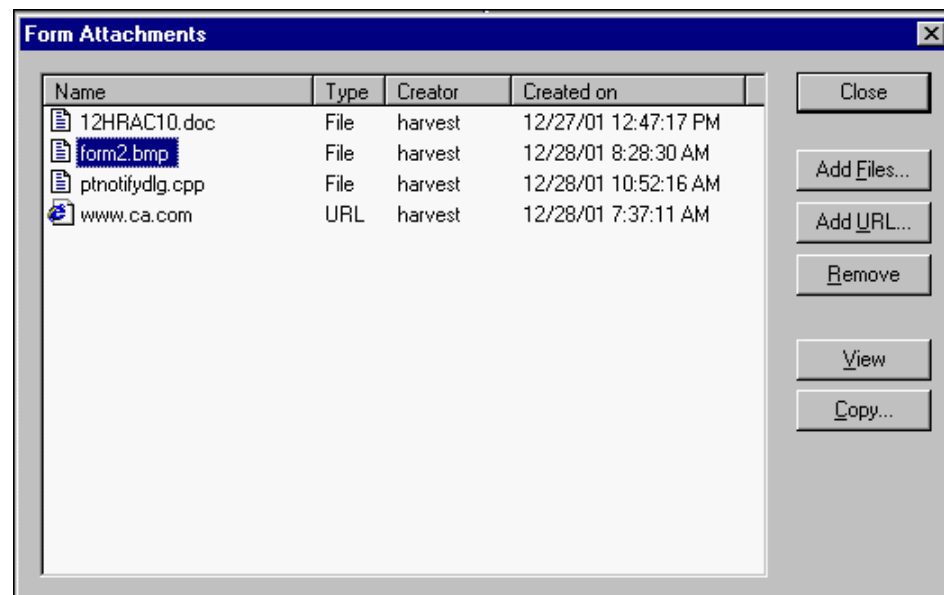
Two types of form attachments exist: file and URL.

- File types include text, doc, and HTML. These files are copied to the Harvest database. Files can originate on the local machine, a network drive, or a remote Harvest agent filesystem.
- URLs reference web sites. Only the URL name is saved in the database.

The Form Attachments dialog lists existing form attachments and allows you to add, remove, view, and copy attachments. To access this dialog, click the paper clip icon button located on the lower left-hand corner of the Form Viewer.



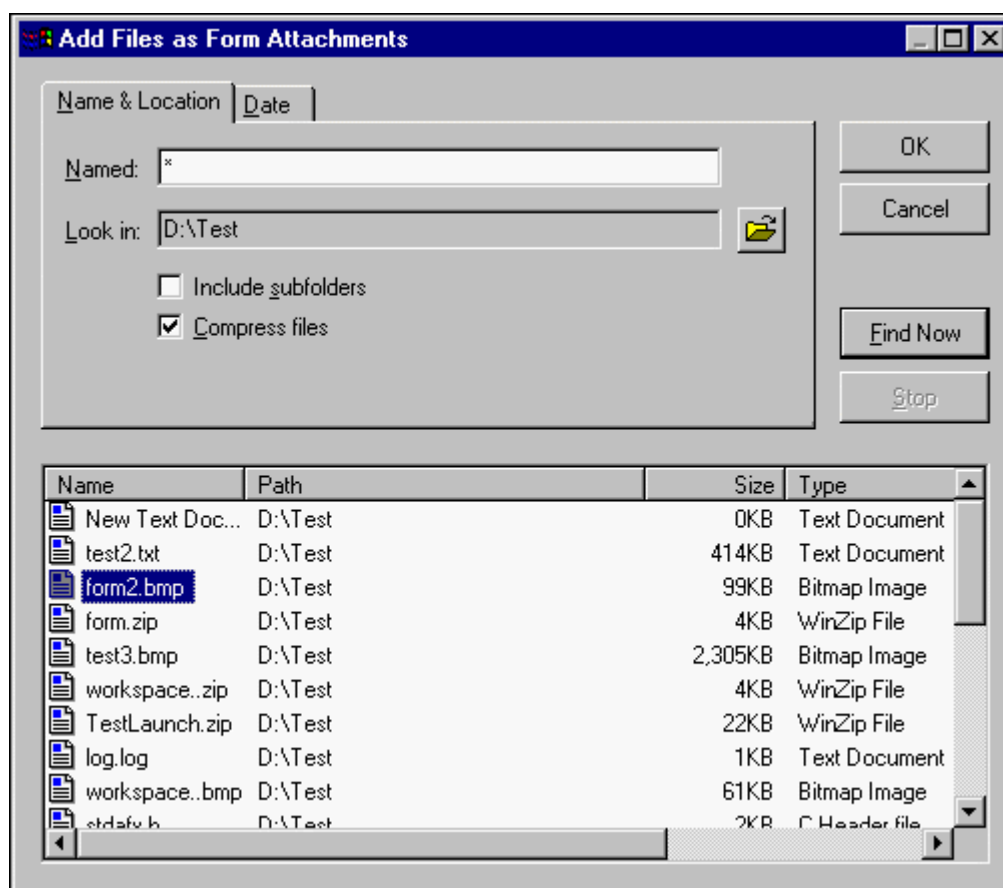
The Form Attachments dialog appears in the foreground.



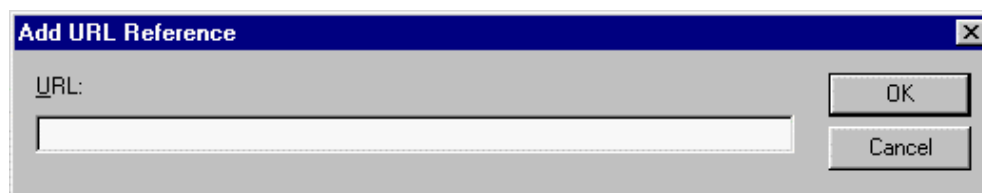
Adding Attachments

Clicking the Add Files button on the Form Attachments dialog, opens the Harvest Find File dialog, which allows you to locate and select files from the local file system, network drive, and remote agent to add to the form. By default the Compress files option is enabled; this compresses files before adding them as attachments. After clicking the OK button on the Find File dialog, your additions are attached to the form.

Deselect the Compress files option when adding files that are already in compressed format. Use of the compress option on a file that is already in compressed format will not reduce the size of the attachment and may actually increase it. Compressed files usually have one of the following extensions: .cab, .gz, .zip, .jpg, .gif, .asf, .ram, .mp3, .wav.



Clicking the Add URL button on the Form Attachments dialog opens the Add URL Reference dialog, in which you can type a URL address to reference a web site from the form. After clicking the OK button on the Add URL dialog, your additions are attached to the form.



Removing Attachments

To remove an attachment from the list of attachments associated with the current form, you can select the attachment on the Form Attachment dialog list box, and then click the Remove button. This does not delete the selected attachment; it simply removes the association.

Viewing Attachments

Clicking the View button on the Form Attachments dialog displays the selected file or URL.

- Text or document files, graphics and other files display in their associated applications.
- URLs display in the Internet Explorer.

Copying Attachments

Clicking the Copy button on the Form Attachments dialog, allows you to copy the selected file to the local file system, a network drive, or a remote file system. After clicking Copy, the Select a Directory Path dialog opens and you can locate a destination for the attachment copy. Clicking the OK button executes the copy. The file is marked read-write.

Note: You cannot copy URL attachments.

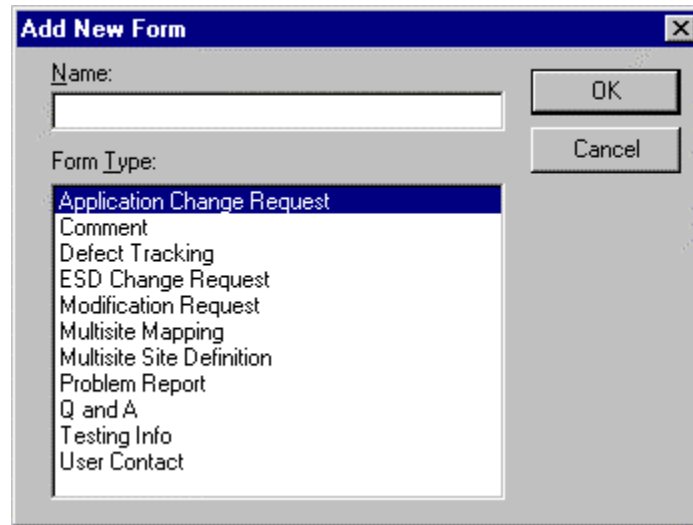
Package and Form Associations

Forms attain their greatest usefulness through association with packages and can only be created in association with one or many packages. Together, they represent one unified whole: a problem or enhancement request and the changes made in response to it. Each package can have one or more forms associated with it, linking the package to the change tracking features of Harvest.

You can associate a form with a package in three ways:

- One form can be associated with a package by defining the Create Package process to automatically create a form when executed. The type of form to be associated with the package is defined by the administrator when the process is set up.
- You can associate a form with a package by using the Forms tab of the Package Properties dialog.
- You can add and associate a form with a package by right-clicking the package and choosing Add New Form from the shortcut menu to open the Add New Form dialog, shown below.

Add New Form Dialog

The image shows a Windows-style dialog box titled "Add New Form". It has a standard title bar with a close button (X). The dialog is divided into two main sections. The top section is labeled "Name:" and contains a single-line text input field. To the right of this section are two buttons: "OK" and "Cancel". The bottom section is labeled "Form Type:" and contains a list box. The list box has a blue header bar and contains the following items: "Application Change Request" (which is currently selected and highlighted in blue), "Comment", "Defect Tracking", "ESD Change Request", "Modification Request", "Multisite Mapping", "Multisite Site Definition", "Problem Report", "Q and A", "Testing Info", and "User Contact".

Name—Enter a name for the new form.

Form Type—Select a form type from the list and click the OK button to create a form and associate it with your current package.

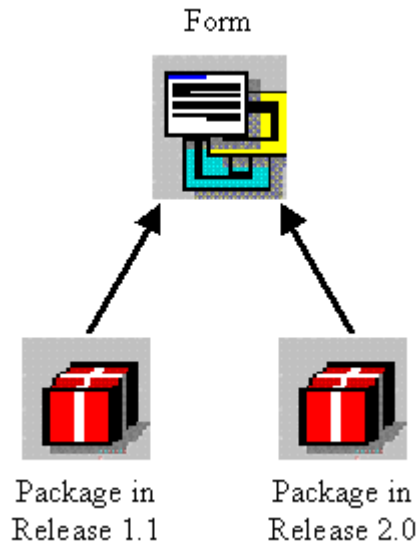
Harvest allows you to create the following associations between forms and packages. All associations are bi-directional, enabling you to view an associated form from a package, or an associated package from a form.

Note: Deleting a package deletes its associated forms if those forms are not associated with another package. However, deleting a form does not delete any associated packages.

One Package to One Form—This association is the simplest relationship between forms and packages. It is commonly used when one form represents the description of, and one package represents the solution to, one problem.

One Package to Multiple Forms—When one package is used to resolve several related problems, multiple forms can be associated with it. This relationship is also useful when more than one kind of information needs to be recorded for a package. Comment, Testing Info, and Problem Report forms can be associated to include comprehensive information about each package.

Multiple Packages to One Form—This kind of association can be used to link changes made to fix the same problem in two projects. For example, if a critical problem is reported, it can be assigned to a maintenance release of an application (Release 1.1) at the same time it is assigned to the current long-term development release (Release 2.0). A package is created in each project and linked to the same Problem Report form. Both packages share the same form, rather than separate copies of it. If the form is modified, both development groups can immediately see the changes.



Form and package associations can be changed at any time. For example, several packages with associated forms are created addressing separate issues. Subsequent analysis reveals that these separate issues are all caused by one underlying problem. At this point, all the forms can be associated with one package, and the remaining packages can be deleted.

From the Projects tab, you can list a package's associated forms by clicking the plus (+) sign next to the package.

Package Movement Through the Life Cycle

Packages normally travel sequentially from one state to the next in a life cycle. Returning packages to a previous state also usually occurs sequentially. For example, consider a simple life cycle with four states: Development, Unit Test, QA, and Release. The following promote processes would move a package sequentially through this life cycle:

Development to Unit Test

Unit Test to QA

QA to Release

Corresponding demote processes would return a package sequentially:

Release to QA

QA to Unit Test

Unit Test to Development

This sequential movement of packages is the most simple and straightforward way to manage changes; however, you might want to demote a package across multiple states.

Using the current example, if QA finds a problem, it would be simplest to return the package directly to Development for additional work, rather than demoting it to Unit Test and then Development. This is possible as long as both Unit Test and Development share the same view.

Consider what would happen if Development and Unit Test had different views. A package is promoted from Development to Unit Test and then to QA, updating each view with the changes. Then the package is demoted from QA to Development. The demote process removes the changes from QA, but they still appear in Unit Test, although the current state of the package is Development.

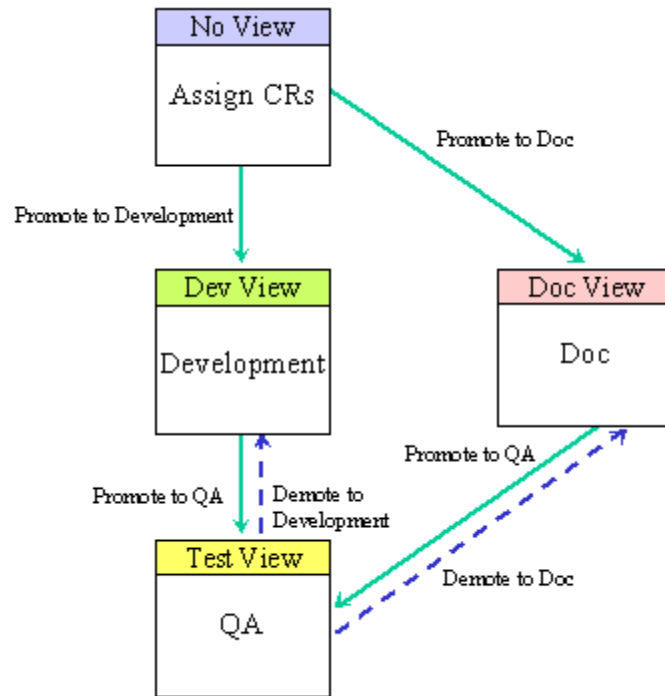
Skipping states in the life cycle can create a confusing situation in which package changes do not appear in the view associated with the state that the package skipped.

If Development, Unit Test, and QA all have their own views, a similar situation could occur if you promote a package from Development to QA. The promote process would update QA with the package changes and change the state of the package to QA, but it would not update Unit Test.

Note: To avoid this confusion, follow this rule: Packages should always move forward or backward sequentially through the life cycle from one state to the next, unless two or more states share the same view.

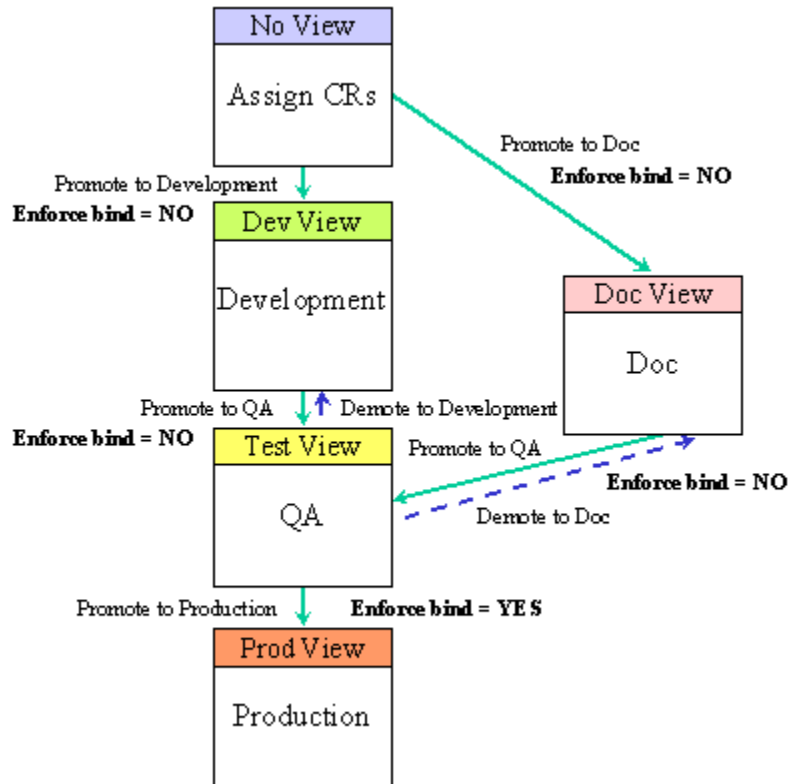
Packages can move to more than one next state if multiple promote processes have been set up. This setup can be used to move packages along different paths.

The following figure shows how users in the Assign CRs state can send a package to either Development or Doc, depending on whether the CR pertains to software or to documentation. Packages in both Development and Doc are then promoted to QA for quality assurance testing and can be demoted if the packages need more work.



If the Enforce Package Bind option is not set by the administrator, you can decide how you want a package to move through the life cycle, even if the package belongs to a bound package group. By enabling the Enforce Package Bind option on the Promote Process Execution dialog, all packages belonging to the group must be promoted together.

Typically, gathering packages into one state becomes important during the final stages of the life cycle. The following figure illustrates how you might not enforce package binding during the early stages of the life cycle to allow flexibility of package movement, and then later to restrict the movement, enforcing package binding. Notice in the figure, the enforce package bind option is only enabled during promotion from QA to Production. This gathers the packages into their bound package group in the final state, thus ensuring that Production has all necessary packages for the group.



Processes

This chapter describes in detail the execution of each Harvest process. The process name displayed in the menu and in the dialog's title bar was specified in the properties dialog for the process. For information on the properties dialogs for each process, see the *Administrator Guide*. For information about executing processes using the Harvest web interface, see the help documentation on the web interface.

Process dialogs have three execution buttons.

- **OK** executes the process, closes the dialog, and the results of the execution are displayed in the output log.
- **Apply** executes the process but does not close the dialog; your selection criteria is retained.
- **Cancel** closes the execution dialog. If you have made any selections in the dialog, they are not retained.

To invoke a process execution dialog:

1. Right-click the object and choose *Process_name* from the shortcut menu.

Approve

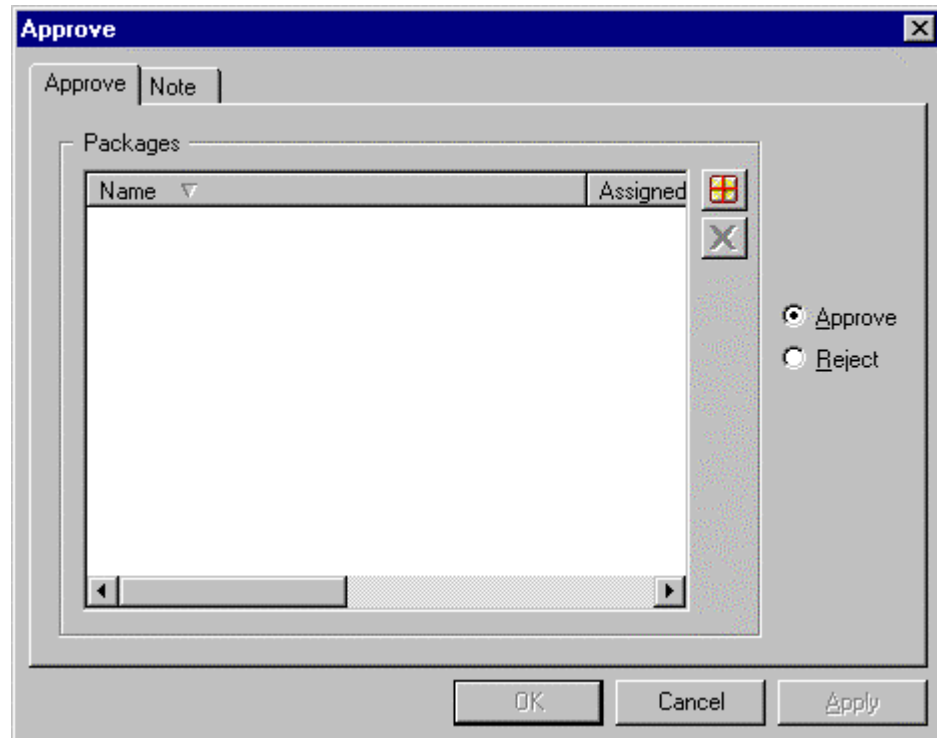
The approve process allows designated users to approve or reject a package so that it can be promoted or moved. When more than one approval process is defined in a state, the approval processes are treated as alternative to one another, so a package can be promoted or moved as soon as one approval is complete.

A process can require the approval of both individual users and user groups. If the approval of a group is required, only one user in the group needs to execute the approval process. If a member of a group rejects a package, that user must approve the package to remove the rejection; the approval of another user in the group does not override it.

The approval of a user might be required in addition to that of a user group to which the user belongs. When this user approves a package, both requirements are met in one execution.

If a package is part of a bound package group, all packages in the group must be approved together. Approval or rejection of a package group results in an approval or rejection of all packages that are included in the package group in the state the approval or rejection is applied.

Linked processes execute before and after the approval process completes successfully, successfully means all packages are approved. If the approval of any selected package fails, linked processes do not execute.



Approve and Reject—You can approve or reject packages by choosing the Approve or Reject button. Approve is the default.

Packages—If you select one or more packages on the workbench and then invoke the approve process, the package names populate the Packages list box. If a package belongs to a bound package group, all packages belonging to the group are listed.

To select additional packages to approve or reject, click the button next to this field to open the Select Packages dialog from which you can select a package in the current state to merge.

To remove packages from the approval list, select packages from the list box and then click the Remove (X) button.

Note—In the Note field, an informative note can be specified describing the reasons for approval or rejection.

Check In

The check in process allows you to copy files from the client directory system into Harvest, creating new items or updating existing items that have changed. To check in an existing item, it must be reserved by a package. If the item was originally checked out for concurrent update, the check in process updates the current package's branch. If the item was checked out for update or reserve only, the check in process updates the trunk for the item.

You can check in files from remote locations. For example, you can check in files on an AIX platform into a Harvest repository that is managed by a server process that runs on a Solaris platform where the file cannot be accessed through NFS. Access to data files in remote locations requires a user name and password for the remote machine.

When the properties for a check in process are defined, defaults are specified for many of the process options. These defaults determine how the process execution dialog appears when it is first accessed. Users can override the defaults.

Checking In New Items

New items can be added to the repository using the check in process. The following rules indicate how Harvest manages the checking in of new items:

- When a new item is checked in it appears as a version 0, the same as items that are added to a project using the configure baseline operation.
- When a new item is checked into a view path location a version 0 is created. Checking in the same item to a different view path location within the same project creates a new version (version 0).
- If new item and item paths are created in the repository by using the Preserve and Create Path Structure option of the check in process, the item paths are immediately visible in all working views of the project. The items in the view paths are initially visible only in the working view to which they were checked in.

File Conversion

When sharing files between UNIX, PC clients, file conversions can be required during check in. The end of line and end of file markers conversions are performed in the files marked as text as specified in the Text File Extensions tab. This tab is accessed through the Repository Properties dialog.

File and Item Case-Sensitivity

Harvest check in behavior differs between UNIX and PC platforms because of the case-sensitive/case-aware behavior of file systems on these platforms.

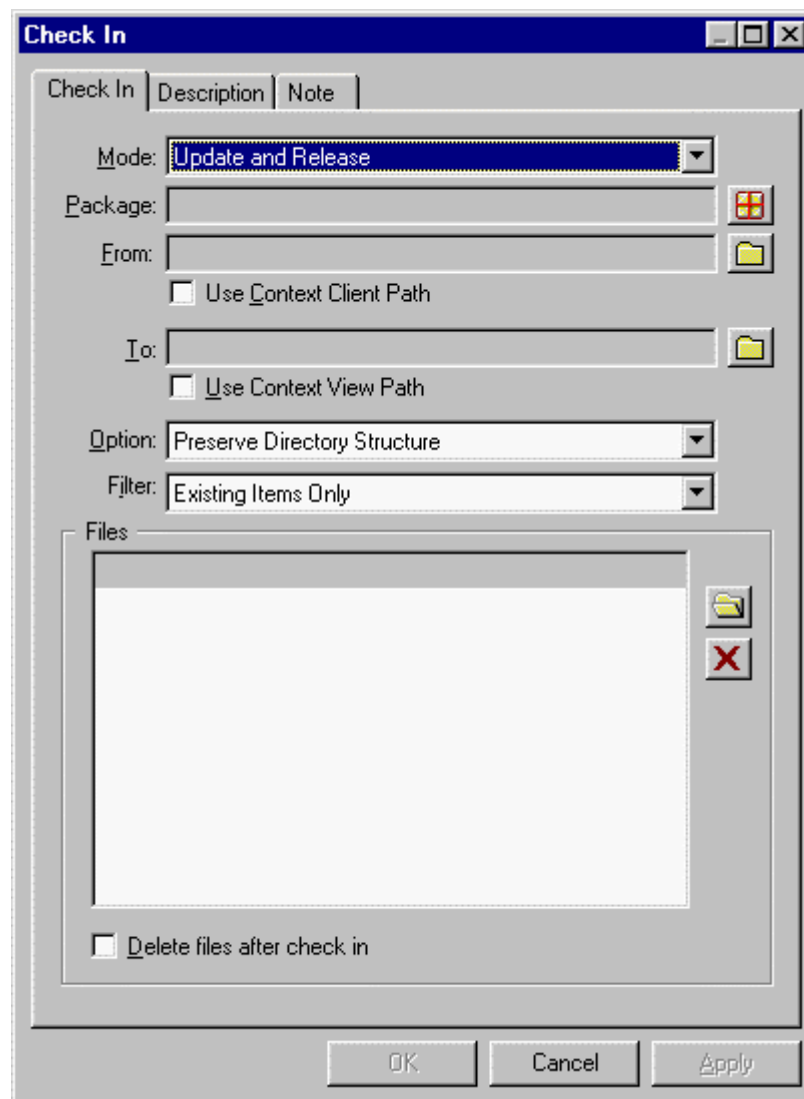
Support both case sensitive, case-insensitive, and case-aware file systems. Must be able to handle case-sensitivity issues based on platform. For example the file File.c is not the same as FILE.C on a UNIX platform but is the same for a PC or MVS platform.

On UNIX clients, Harvest is case-sensitive for file and item names. If the item file.c is checked out to a UNIX client, the file must be checked in with this name; if the case of the name is changed to FILE.C on the client system prior to check in, it is checked in to the repository as a new item.

On PC clients, Harvest is not case-sensitive. If the item File.c is checked out to a PC client, this item can be successfully checked in as FILE.C, file.c, or as any other case. Harvest is, however, case-aware on PC platforms, and the repository always maintains the case of the item as it was originally checked in or loaded.

Signature Files

Checking in an item automatically updates information in a signature file maintained by Harvest in each directory containing CCM-related files.



Mode—A file cannot be checked in unless its corresponding item is reserved by the package performing the function. The only exception to this is items being checked in for the first time. The check in function operates in one of three modes, governed by the state of the Mode drop-down list:

- **Update and Release**—The new item or version is checked into the repository and, if the item was reserved by a package during a previous check out, the reserved tag is removed from the item. The permission on the client file is automatically set to read-only so that changes cannot be made to it until it is checked out again.
- **Update and Keep**—The item in the view is updated (or created) and the current package keeps it reserved, and the file permissions unchanged, so that more updates can be made.

- **Release Only**—No check in is performed and the item is not updated, but the item is no longer marked as reserved for the current package. The permission on the client file is automatically set to read-only so that the file cannot be changed until it is checked out again.

Check in for Release Only does not require the local file system file to exist. If the file does not exist on the local file system, you need to right-click the reserved version and then choose the check in process.

Note: Selecting Release Only mode implies an item filter of Existing Items Only because this operation requires existing reserved items in the repository; the Filters drop-down list is disabled.

Package—Any updates made to an item are associated with the package selected for the Package field. If you select a package before choosing the check in process, the package name populates the Package field. If you do not have a package selected before invoking the check in process, you can select a package by clicking the button next to the Package field. This opens the Select a Package dialog, which allows you to locate and select packages.

From and **To**—These fields work together to help you synchronize the internal (repository) and external (file system) structures. They are typically used to establish a point at which paths in the repository mirror working directories on the client.

From—Files are checked in from the client file directory. The client path displayed in this field is determined in one of these ways:

- If you use the Find File dialog to select files, the most common path in the file list populates this field. When your file list changes, the field is updated to display the most common path.
- If you select a file on the Files tab of the workbench and execute the check in process, that context is used in the execution dialog, overriding the Default Context dialog setting if it differs.

Use Context Client Path—If you select this option and you used the Default Context dialog to set a default client directory location, the file path will populate this field.

To—The path selected for the To field specifies the location in the destination view for the files being checked in. To change the path, click the button next to this field. This opens the Select a Repository Item Path dialog that allows you to browse through available paths and select a location.

If the files being checked in include a directory specification, as would be the case during recursive check in, this directory specification is appended to the view path specified here when either of the preserve structure check in options is being used.

Use Context View Path—If you select this option and you used the Default Context dialog to set a default view path, the view path will populate this field.

Option—The Options drop-down list provides a variety of ways to check in files and filter the files being checked in. Select **one** of the following:

- **Preserve Directory Structure** checks in the selected files to repository view paths with names that correspond to their client directory location, if these view paths currently exist.
- **Preserve and Create Path Structure** checks in selected files to paths with names that correspond to their client directory location, and creates any view paths that do not currently exist.
- **All Files to Same View Path** checks in all selected files to the same path in the destination view, ignoring the client directory structure.

Filter—The Filter drop-down list provides a variety of ways to filter the files being checked in. Select **one** of the following:

- **New or Existing Items** checks in all selected files, if they are reserved by the package or did not previously exist.
- **New Items Only** limits the check in to files that do not have corresponding items in the Harvest repository.
- **Existing Items Only** limits the check in to files that have corresponding items reserved by the package. Any files without corresponding items are skipped. This filter can be used to prevent the existence of unwanted files, such as temporary files or templates, in your repository.

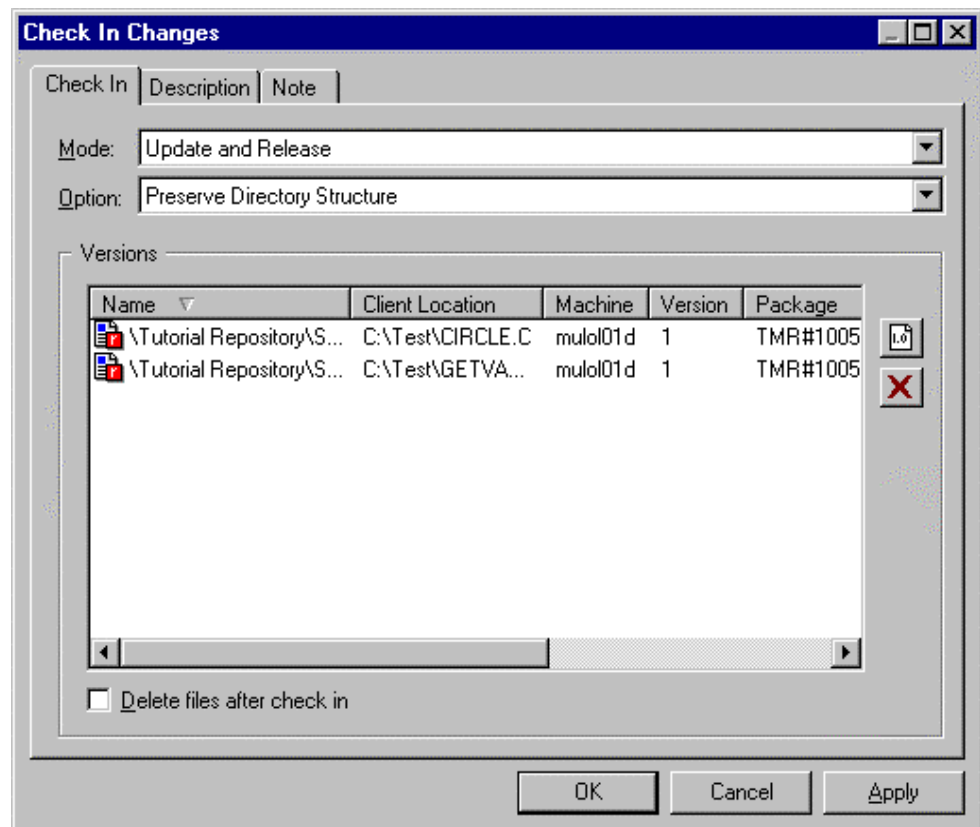
Note: Only the item filters enabled in the Check In Process Properties dialog are available for selection in this field.

Files—If you select files in the client directory before invoking the process dialog, the files populate the Files field. Files can also be selected by clicking the button next to the Files field. This invokes the Find File dialog, which enables you to use multiple filtering operations to let you search precisely for files and to navigate through directories available on your system to select a file.

Delete files after check in—This option deletes from the client file system all files that are successfully checked in with the mode Update and release, or Release Only. Deleted files are also removed from the signature file (harvest.sig) on the client.

Package Check In

If you right-click a package and choose a check in process from the shortcut menu, and that package has associated versions, the package check in process execution dialog is invoked. The package check in dialog does not allow you to specify the view path or client path. The paths you used during check out are retained and used for the package check in. For example, you could associate various versions from different view paths to one package and then check out the package to your client directory. After updating the versions, executing a package check in would place the changed versions in their original view path locations.



Command Line Check In

On the command line, you can specify values for all the check in and check out command options on the command line. Harvest uses the signature file to streamline the operation of command line check in (hci) and check out (hco) and to prevent you from making repetitive entries.

Four attributes of a file are maintained in the signature file that can be used with hci and hco. When an item is checked out from Harvest, information about its context – including project, state, view path, and package – is maintained in the signature file. All of these parameters are required to check in the item. If these parameters are not specified on the command line, Harvest searches the signature file for them and uses the signature attributes. This makes it very easy to check a file back into its place of origin.

The same information is maintained in the signature file during check in and can be used by hco to determine where to find the item to check out.

To make the usage of the signature file even more complete, you can specify four directory attributes with the signature file update (hsigset) utility: project, state, view path, and package. Then, if information for a file is not available in the signature file, the directory attributes can be used by hci and hco. For more information about hsigset, see the *Reference Guide*.

If multiple files are selected for the check out or check in operation, all of the files must have the same signature file attributes (project, state, package, and view path). The operation fails if any selected files do not match the specified attributes.

The following table summarizes the way that Harvest searches for context values for hci and hco command options.

Source	Comment
Command line	The command line arguments have the highest level of priority. Any argument specified on the command line overrides other values for a context attribute.
File's Signature	If no command line argument is present, Harvest searches the signature file for information about the file. For example, if the file was checked out for Update by package STR-0432 and no package value is specified on the command line, Harvest uses the value of STR-0432 for the check in. For the check out or check in operation to execute successfully, all files must have the same signature file information (project, state, package, and view path).
Directory's Signature	If a context attribute is not available for a file, Harvest searches the directory attributes. For example, you might have set the view path attribute for the directory. You can check out new items that have not previously been checked out without specifying a view path on the command line and Harvest determines their location in the repository.

The order of events described above is different when the recursive mode of check in/check out is specified. For recursive operations, if values for project, state, package, and view path are not specified on the command line, Harvest then looks for these values in the directory attributes portion of the signature file.

Default Values

Harvest bases the execution of a command line check in on a combination of the options either supplied or defaulted. Once options are specified either through the command line or the signature file, other values can be generated by default. The default values include the following:

- The check in process. If no process name is specified, hci executes the first check in process in the specified state.
- Process modes and access. The definition of the specified check in process in Harvest determines the default check in mode, as well as the access to the process.
- The item filters (new or existing, new only, and existing only). If no option is specified on the command line, the default value of the selected process is used.
- The check in path option (-op). If no option is specified on the command line, the default value Preserve Directory Structure is used.

- The output file (-o). Output is automatically generated to the default output device if no -o value is specified on the command line.

Example

```
hci -b kauai -en 'Release 2.0' -st development -p STR-0023-015  
-vp /Rep1/common/src/ -ur *.c
```

In this example, a user is checking in all files with an extension of .c from the current working directory to the view path /Rep1/common/src. The context is defined by the Release 2.0 project and development state. Because no process name is specified, the first check in process in the specified state is used. The package being used is STR-0023-015. The check in mode is Update and Release. The user is prompted for a change description, because the -de option is not specified.

Check Out

The check out process allows you to copy versions of items under Harvest's control to external directories where the items can be updated or browsed. Alternatively, items can simply be reserved without actually copying the data to a client directory.

You can check out an item to a remote location. Access to data files in remote locations requires WRITE privilege based on the registered user name and password for the remote machine.

When the properties for a check out process are defined, defaults are specified for some dialog options. These defaults determine how the process execution dialog appears when it is first accessed. Each user can override the defaults.

Signature Files

Checking out an item automatically updates information in a signature file maintained by Harvest in each directory containing CCM-related files. Signature files form the basis for the Synchronize mode of check out.

Rules

The following rules govern the check out process:

- **Access.** To check out an item for Browse or Synchronize, you need at least view access to the item. To check out an item for update or concurrent update or to reserve an item, you need edit access to the item.

- **Duplicates.** Any version of an item can be checked out, not just the latest. You can check out only one version of an item at a time into the same directory on the file system.
- **Package.** To check out an item for update, concurrent update, or reserve only, a package name in the current state is required. A package can only create one reserved version per item, either on a branch or on the trunk.
- **Version.** The only trunk version that you can check out for update is the latest. It can only be checked out if it is not already reserved by another package. This version is available for updating even if it does not yet appear in the current view. To check out an earlier trunk version for updating, the concurrent update mode of operation must be used.
- **Updating a Branch.** To check out and modify a branch version, you can use either the update or concurrent update mode of check out. Both create a reserved branch version that is replaced when the file is checked in. The package specified for the check out must be the same as the package that created the branch.
- **Unmerged Branches.** A package can only have one unmerged branch version per item. A second attempt to check out an item for concurrent update fails if an existing unmerged branch is found for that item with that package. The current branch version should be either merged or deleted before attempting to use the concurrent update mode of check out again for the same item with the package.
- **Reserve Only.** The reserve only mode of check out acts like check out for update, but does not copy the version to the client directory. The reserved version is tagged as reserved (R).
- **Tagged Versions.** Versions that are tagged as merged (M) or removed (D) cannot be checked out at all. A reserved version (R) can only be checked out for Browse mode; its content is the same as the previous version.

File Permissions

Harvest assumes a long-term relationship between a set of user working directories and a Harvest repository structure. Within these working directories, Harvest uses file permissions to indicate files that are not available for update. File permissions are altered by Harvest during check out and check in.

WARNING! Because Harvest manages these permissions according to the state of corresponding items in Harvest, it is recommended that users do not alter file permissions. This could lead to unexpected results during check out or check in.

File permissions are handled differently depending on the client's operating system. For example, UNIX makes much greater use of file permissions than PC operating systems. On Windows platforms, Harvest uses only one level of access permission, represented by the read-only attribute.

- On PC client systems, checking out a file in Browse or Synchronize mode creates a file on the client file system that is marked as read-only. This indicates that the file cannot be modified and checked in because these modes do not create a reserved version in Harvest. When checking out for Update or Concurrent Update, the file is given normal (write access) file status.
- On a UNIX-based system, when a file is checked in to Harvest, information about file permissions is maintained in the Harvest database. When the corresponding item is checked out for Update, the same file permissions are set on the client file, with one exception. Any time an item is checked out for Update, the user performing the update is always given write access to the file so that changes can be made.

For example, consider a UNIX file with the following permissions:

```
r-xrwxrwx
```

When this file is checked in to Harvest, the file permissions are stored. Upon check out for Update, the permissions are:

```
rw-rwxrwx
```

When a file is checked out for Browse, Harvest does not use the exact information it stored with the file. It places the file in the directory without write permission for user, group, and other. The permissions now look like this:

```
r-xr-xr-x
```

File Conversion

When sharing files between UNIX and PC clients, file conversions might be required during check in. The end of line and end of file markers conversions are performed in the files marked as text, as specified in the Text File Extensions tab of the Repository Properties dialog.

When sharing files between UNIX/PC and MVS, the ASCII/EBCDIC character conversion can be done. The rules to activate this conversion are:

1. If the repository containing the file is configured to share both MVS and UNIX/PC platforms, a mapping mechanism is used to create an extension for the file. The resulting extension is used to determine the file type in the Text File Extensions tab.
2. If the repository containing the file is configured for MVS users only, the file is converted if it is not a binary file.

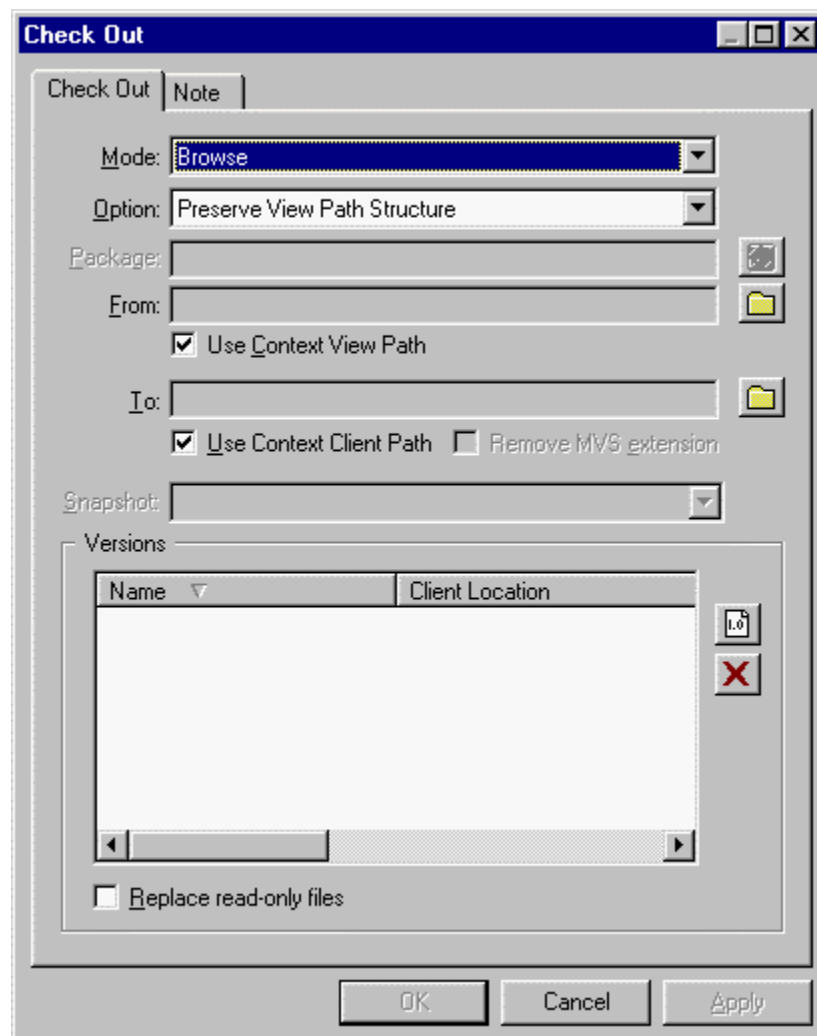
File and Item Case-Sensitivity

Harvest behavior differs between UNIX and PC platforms because of the case-sensitive/case-aware behavior of file systems on these platforms.

- On UNIX clients, Harvest is case-sensitive for file and item names. If the item file.c is checked out to a UNIX client, the file must be checked in with this name; if the case of the name is changed to FILE.C on the client system prior to check in, it is checked in to the repository as a new item.
- On PC clients, Harvest is *not* case-sensitive. If the item File.c is checked out to a PC client, this item can be successfully checked in as FILE.C, file.c, or as any other case. Harvest *is* case-aware on PC platforms, and the repository always maintains the case of the item as it was originally checked in or loaded.

File Date and Time

Although Harvest stores the file modification date and time, the date and time associated with the file created when an item is checked out reflects the current date and time. This algorithm is followed because it provides better support for building an application by ensuring that the source files are more recent than object files.



Mode—The options on the Mode drop-down list vary, depending on how the check out process was defined during setup. One or more of the following options appears; select **one**:

- **Update** copies selected versions to the destination client directory and creates a reserved version on each item's trunk, allowing the corresponding files to be checked back in. The permission on a read-only file is changed to normal (write access) when this mode of check out is used.
- **Browse** copies the items to the destination directory but does not allow you to check the files back in. The read-only attribute is set on files checked out for Browse mode.
- **Synchronize** determines the versions of the files in the client file system using the signature file, and selected versions are checked out only if they differ (either older or newer) from those in the external file system. Items are checked out in a read-only mode. The check out for Synchronize mode is especially useful in the build process.

- **Concurrent Update** copies selected versions to the destination client directory and creates a reserved version on a package branch for each item. All package updates accumulate on this branch. The permission on a read-only file is changed to normal (write access) when this mode of check out is used.
- **Reserve Only** does not move any data to external directories but creates a reserved version with a reserved tag (R) on each item's trunk so that corresponding files can be checked in.

Option—The check out options work in conjunction with the Recursive search option on the Select Version dialog. Normally, only items in one path are displayed in the dialog. When Recursive search is selected, Harvest searches all paths beneath the current path and displays the item versions matching the other filtering criteria being used on the dialog, allowing you to select items from multiple paths for check out. Select **one** of the following:

- **Preserve View Path** checks out all selected items into corresponding client directories, if they already exist. If the directories do not exist, an error message is displayed and the items are not checked out.
- **Preserve and Create Client Directory** checks out all selected items into matching client directories, and creates any client directories that do not currently exist.
- **All Items to Same Client Directory** places all selected items directly beneath the specified client directory, ignoring the path structure within the repository.

Package—The Package field is active when the check out mode is update, concurrent update, or reserve only. The reserved version placeholder created by this check out operation is associated with the package selected for this field. If you select a package before choosing the check out process, the Package field is automatically populated with the package name. If you do not have a package selected before invoking the check out process, you can select a package by clicking the button next to the Package field. This opens the Select a Package dialog that allows you to locate and select packages. If only one package exists in the current state, this package is selected by default for this field.

From and **To**—These fields work together to help you synchronize the internal (repository) and external (file system) structures. They are typically used to establish a point at which paths in the repository mirror working directories on the client.

From—The view path displayed in this field specifies a location in the repository from which items are checked out. The view path displayed in this field is determined in one of these ways:

- If you use the Find Version dialog to select versions, the most common path in the version list populates this field. When your list changes, the field is updated to display the most common path.
- If you select a version on the Projects tab of the workbench and execute the check out process, that context is used in the execution dialog, overriding the Default Context dialog setting if it differs.

Use Context View Path—If you select this option and you used the Default Context dialog to set a default view path, the view path will populate this field.

To—The directory selected for this field specifies the destination on the file system for the checked-out files. If either of the preserve structure check out options is being used, the check out specifies a path and this specification is appended to the file system directory.

If the files being checked out include a directory specification, as would be the case during recursive check out, this directory specification is appended to the client path specified here when either of the preserve structure check in options is being used.

Use Context Client Path—If you select this option and you used the Default Context dialog to set a default client directory location, the file path will populate this field.

Snapshot—The Snapshot drop-down list enables you to select a snapshot from which you can select versions to check out. This field is enabled when a snapshot exists in the view in which you are executing the check out process.

After choosing a snapshot, click the Select Version button located next to the Versions list box. This opens the Select Version dialog enabling you to select versions from the snapshot and return them to the Check Out Process Execution dialog.

Versions—The Versions list box lists the versions you have selected for check out. If you have selected a version before choosing the check out process, the version appears in the list box. If you do not have a version selected before invoking the check out process, you can add one or more versions by the plus sign (+) button next to the Versions list box to open the Select Version dialog. You can remove versions by selecting them and clicking the remove (X) button.

When you select an item for check out, Harvest automatically selects the item's latest versions without tags to populate the check out list box.

When you select a tagged version for check out, Harvest automatically eliminates that version from your check out selection and it does not populate the check out list box. For example, if you select three versions including one that has a reserve tag and then choose check out, only the two untagged versions populate the check out list box.

Replace read-only files—This filter controls whether existing read-only files on the file system should be replaced by the checked-out files. This allows you to replace files that you previously checked out as read-only or checked in. If a corresponding file does not exist on the file system, the item is checked out regardless of the value for this option.

Files that are not read-only on the PC file system, or that have write permission on the UNIX file system, are never overwritten. Harvest assumes that such a file has been checked out for update and not checked back in yet. Overwriting such a file might cause you to lose unsaved changes.

On UNIX systems, file ownership affects the results of this option. When used alone, the Replace Read-Only Files option overwrites read-only files only if the user executing the check out is the owner of the files being replaced.

MVS Check Out

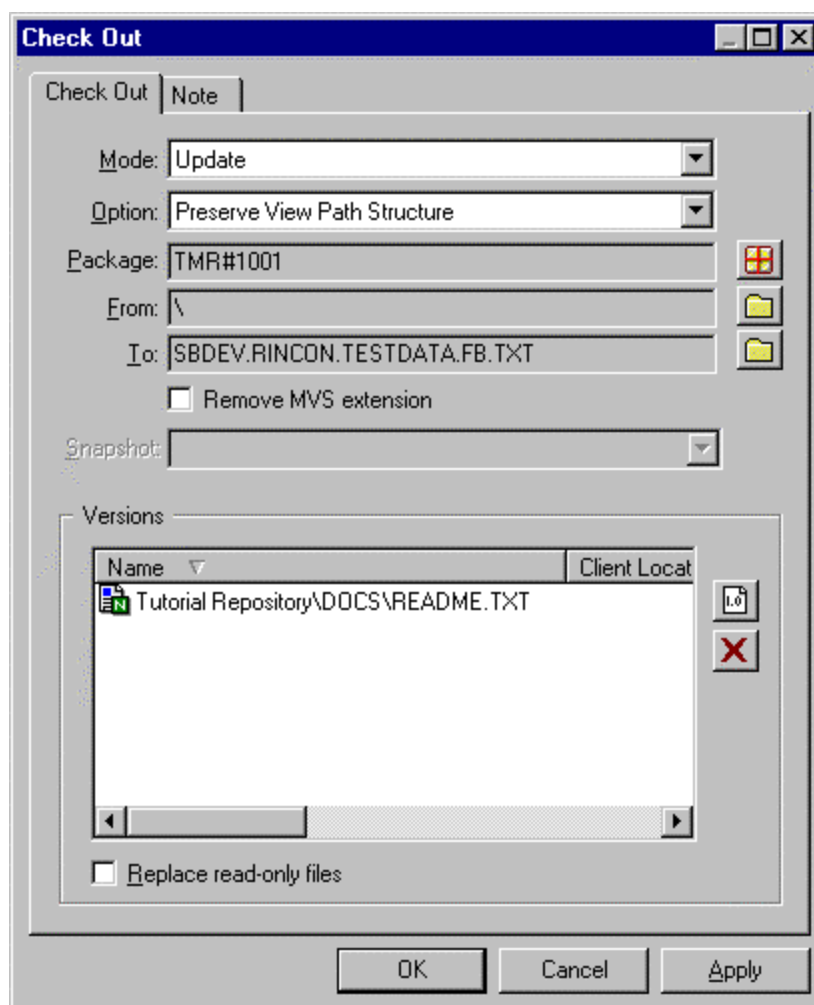
When performing an MVS check out, the check out dialog dynamically changes after choosing an MVS PDS to populate the To field. An additional option is added to the dialog, Remove MVS extension. This option works in conjunction with the To field described below. If an item has an extension and is checked out to a PDS, the item name is the PDS member name and the extension is added to the data set as the last level qualifier. If the data sets already exist with the appropriate last level qualifier, enable the Remove MVS extension option prior to check out.

The agent applies the item extension to the last level qualifier at check out. This option enables you to check out multiple items with different extensions to multiple MVS files on a single check out. If the PDS does not exist, it is created at check out.

The To field destination name reflects the removal.

To—The data set selected for this field specifies the destination for the checked-out files.

Remove MVS Extension—When selected, this option removes the last level qualifier from the selected data set.



Command Line Check Out

You can specify values for all the check in and check out command options on the command line. Harvest uses the signature file to streamline the operation of command line check in (hci) and check out (hco) and prevent you from making repetitive entries. For more information about the signature file and the check out process, see Command Line Check In.

Harvest bases the execution of a command line check out on a combination of the options either supplied or defaulted. Once you specify options either through the command line or the signature file, other values can be generated by default. This includes the following parameters:

- The check out process. If no process name is specified, hco executes the first check out process in the specified state.
- Process modes and access. The definition of the specified check out process in Harvest determines the modes of check out that are available, as well as the access to the process.
- The check out path option (-op). If no option is specified on the command line, the default value Preserve View Path Structure is used.
- The Replace Read-Only Files option (-r). Unless this option is specified with the -r option, this option is set to off and read-only files are not replaced by the check out.
- The output file (-o). On UNIX platforms, output is automatically generated to the default output device if no -o value is specified on the command line. PC platforms do not define a standard output device, so output must be directed to a file. If no output file is specified with the -o option, Harvest creates one named hco.log in the working directory.

If the check out process has other processes linked to it, they are executed during the command line check out. Linked UDP processes cannot write to the standard output when triggered by a parent check out process invoked by hco.

Example

```
hco -b kauai -en 'Release 2.0' -st development -p STR-0023-015 -r  
-vp /Rep1/common/src/ -up file1.c
```

In this example, a user is checking out file file1.c from the view path /Rep1/common/src. The context is defined by the Release 2.0 project and development state. Because no process name is specified, the first check out process in the specified state is used. The package being used is STR-0023-015. The check out mode is Update. Read-only files are overwritten.

Compare Views

The compare views process generates a report showing the differences between any two views, either snapshot or working, which exist in any project. You can select any or all of the following options to generate the report:

- Items that exist only in the first view.
- Items that exist only in the second view.
- Items that exist in both views but have different contents.
- Items that exist in both views but have identical contents.

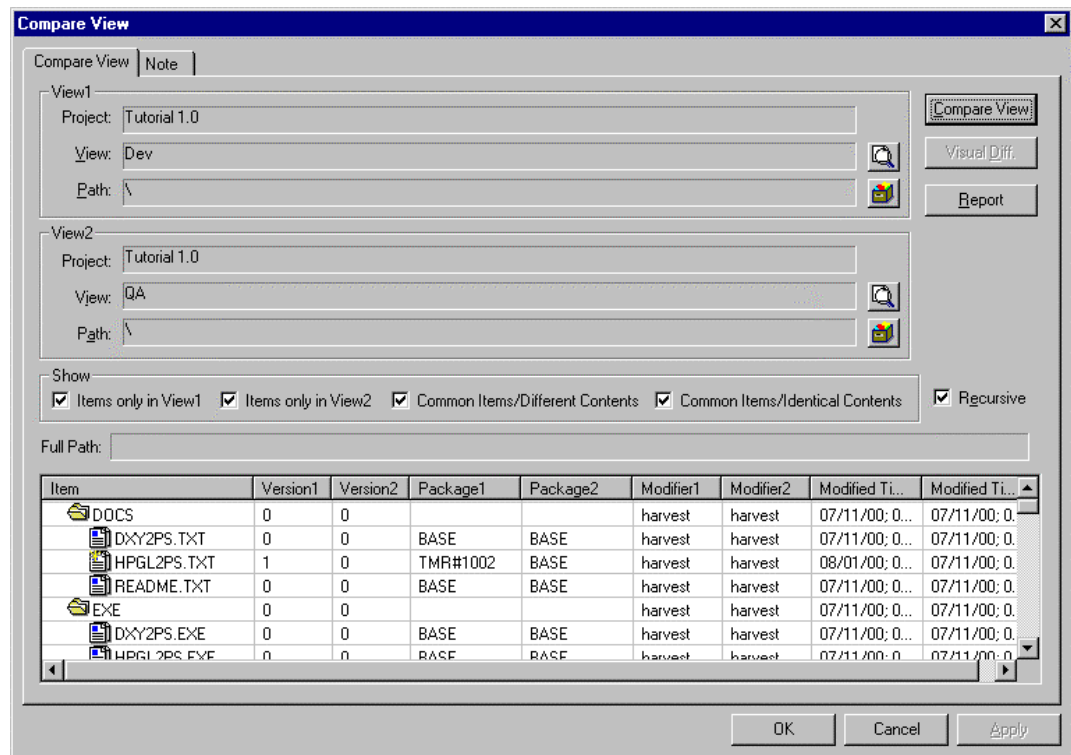
Two options are available for the Compare Views Report:

- Summary gives only the names of the items that have differences between the views, or which are unique to one of the views.
- Detailed gives the same information as the summary report, and also shows detailed line-by-line differences between the items that are different in each view.

Both types of reports are sent to the output log, from which they can be copied to the clipboard or saved to a text file.

The following is an example of a Compare View List Difference Report.

```
----- Begin <Compare Views> Process -----  
List Difference Report  
\Tutorial Repository\SRC\CIRCLE.C,0,1,BASE,TMR#1005,harvest,BobS,12/11/00;  
10:28:51,12/15/00; 08:44:08  
\Tutorial Repository\SRC\GETVAL.C,0,1,BASE,TMR#1005,harvest,BobS,12/11/00;  
10:28:54,12/15/00; 08:44:09  
----- End <Compare Views> Process -----
```



Project—Your current project populates this field.

View—The View fields initially contain the view contexts when the dialog was invoked. Clicking the buttons next to the View fields, opens the Select a View dialog, which can be used to select a view.

Path—The Path fields initially contain the repository paths when the dialog was invoked. Clicking the buttons next to the Path fields, opens the Select a Repository dialog, which can be used to select a repository path.

Specify the items you want to show in the Compare View list. You can select one or more of the options:

Items only in View1—This check box indicates that all items in View1 should be listed.

Items only in View2—This check box indicates that all items in View2 should be listed.

Common Items/Different Contents—This check box indicates that all items that are common to View1 and View2 but have different contents should be listed.

Common Items/Identical Contents—This check box indicates that all items that are common to View1 and View2 and have identical contents should be listed.

Recursive—The Recursive option works in conjunction with the Show options. Normally, only items in the path specified in the View and Path fields and specified by the Show criteria are displayed in the dialog list. When Recursive search is chosen, Harvest searches all paths beneath the current path and displays the item versions matching the other filtering criteria being used on the dialog.

Once View1 and View2 are selected and you have specified your item selection, click the Compare View button to show a list of items meeting your criteria. The list of items shows item descriptions shown in columns.

Note: Tagged versions (reserved, merged, removed) are listed as Changed (C) versions in the scroll list, but they cannot be used with the detailed report or the Visual Difference dialog. An error message is generated in the detailed report for each tagged item.

Selecting an item in the list enables the Visual Diff and Report buttons, allowing you to execute a visual difference or generate a summary report. The Full Path field becomes populated with the path of the item you selected.

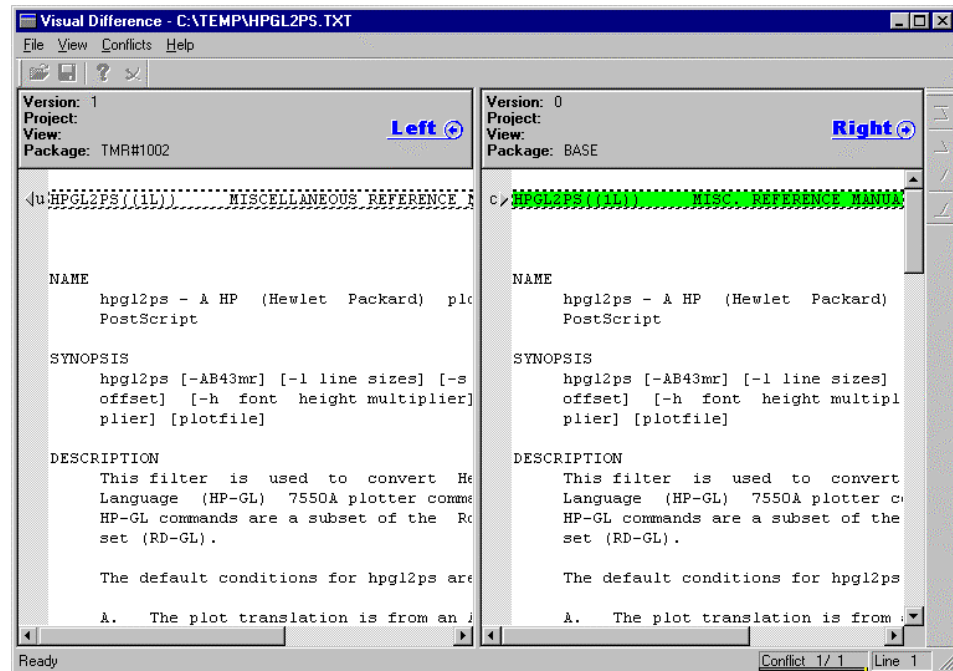
Compare View—The Compare View button displays the items that are unique to one view or which have changes between the views.

Visual Diff—The Visual Diff button is enabled only if an item that is marked as modified in the differences list box is selected. Choosing this button invokes the Visual Difference window, which displays the two versions of the item side-by-side for easy comparison.

Report—The Report button generates a summary Compare Views report, which is sent to the output log. From the output log, it can be printed, copied, or saved.

Visual Difference

The Visual Difference dialog is invoked by clicking the Visual Diff button on the Compare View process execution dialog and displays detailed, line-by-line differences between two files. When the dialog is invoked, the versions being compared are displayed with common blocks (lines the same in both versions) and conflict blocks. Because the Compare View process is read-only, changes cannot be made to the files.



The Visual Difference dialog has two panes: the Left pane and the Right pane. The title bar displays the version names and their contexts.

- **Left Pane**—The Left pane contains the current project and view context when the Compare Views dialog was invoked.
- **Right Pane**—The Right pane contains the comparison view that was selected by the compare views process.

The panes are synchronized to ensure that the panes are displaying the same conflict lines.

Text characters beside lines and color codes indicate the status of the text.

- **u** indicates unchanged text. The color code is navy blue.
- **a** indicates added text. The color code is yellow.
- **c** indicates changed (conflict) text. The color code is green.
- **d** indicates deleted text. The color code is gray.
- If the imaginary symbol option is enabled, a circle indicates the lines that have been added to align the synchronicity of the panes.

Status Bar—The status bar at the bottom of the dialog contains a Conflict meter. The Conflict meter shows your current conflict location and the total number of conflicts. A yellow horizontal bar indicates your location in the sequence of conflicts. The line number of the current conflict is also displayed in the status bar.

Navigation

Conflicting blocks are displayed side-by-side, and the portions of the item that are the same for each version are displayed across the width of the dialog. The Conflicts menu and a toolbar provide you with different options to navigate through the conflicts. The navigation options found in the Conflicts menu are the same as those found on the toolbar. The toolbar can be undocked and relocated on the dialog. The scroll bar on the right side of the dialog can be used to move up and down in the file.

You have the following options:

- **First Conflict** moves you to the first conflict.
- **Previous Conflict** moves you to the previous conflict.
- **Next Conflict** moves you to the next conflict.
- **Last Conflict** moves you to the last conflict.

View

The View menu lets you customize the dialog to show or hide the Visual Difference Toolbar, Toolbar, Status Bar and Imaginary Symbols.

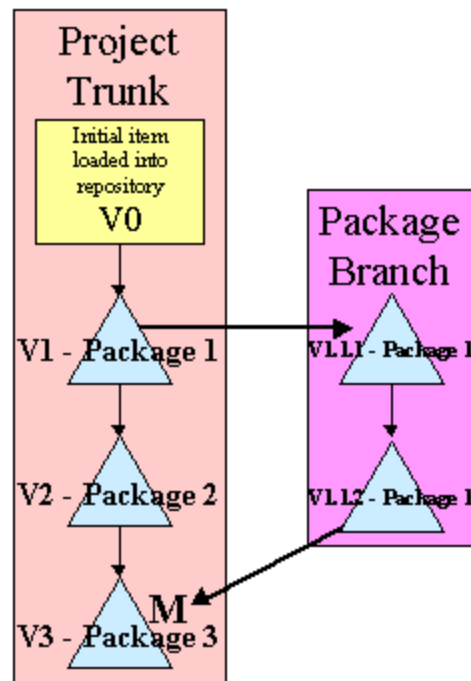
You can double-click on the pane's title bar to hide the title bar and increase your visible text space. Double-clicking again on the top border of the pane restores the title bar.

Concurrent Merge

A version on a branch becomes part the project trunk through the concurrent merge process. Until this process is executed, changes made on the branch exist only on the branch. After all items have been checked in to a branch, invoking the concurrent merge process for the package containing the changes causes the versions on the branch to be combined into a single version on the trunk. The new version created on the trunk is the latest in the view.

If versions exist on the trunk that are more recent than the version from which the branch was created, the version created by the concurrent merge is tagged as merged (M). If the version from which the branch was created is the latest version of the item on the trunk, the version created on the trunk by the concurrent merge has no tag.

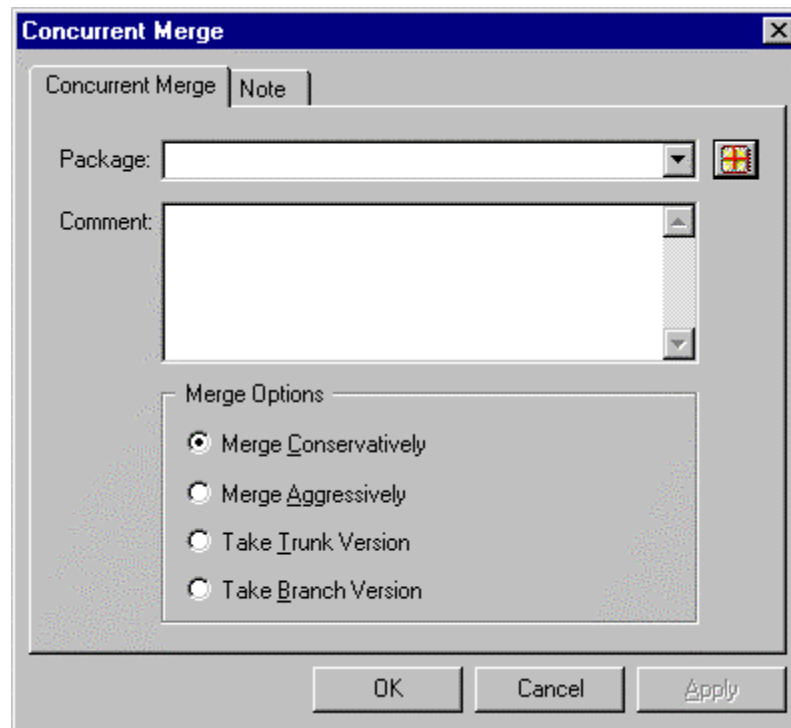
In the next figure, the branch version 1.1.2 is concurrently merged with the project trunk. The version created by the concurrent merge becomes the latest on the view's trunk, version 3, and is tagged as merged.



In the above figure, if version 2 did not exist on the project trunk, the concurrent merge process would have created a non-tagged version 2. Merge tags appear only when changes have been made on the trunk that could conflict with changes made on the branch. If version 1 was the latest version when the merge was executed, there would be no changes on the trunk to conflict with the branch versions, so a merge tag would be unnecessary.

The merge-tagged version of an item must be resolved through the interactive merge process before another version of the same item can be concurrently merged to the project trunk. The check out for update and cross project merge processes also cannot be successfully executed on an item for which a merge-tagged version exists; and packages which contain merge-tagged versions cannot be promoted or demoted to a state in a different view.

Note: If the item is removed after the branch was created, an error is generated during the merge. To merge branch changes, the removed-tagged version (D) must be deleted using the delete version process.



Package—If you have selected a package on the workbench, the selection is copied into the Package field. If not, clicking the button next to the Package field opens the Select a Package dialog from which you can select a package in the current state to merge. When you close the Select a Package dialog, the selected package is displayed in the Package field, allowing you to visually confirm your selection. You can remove packages by selecting them and clicking the remove (X) button.

Comment—In the Comment field, an informative note can be specified describing the reasons for approval or rejection.

Merge Options—Select **one** of the Merge Options buttons to specify how the process executes. During setup the administrator defined the options available.

- **Merge Conservatively** creates a merge-tagged version, regardless of the contents of the versions. An exception is when the branch version is the latest version in the view; in this case, it is closed and a normal version is created.
- **Merge Aggressively** creates a merge-tagged version only when conflicts are found. If no conflicts are found, the branch and trunk versions are merged to create a normal version.

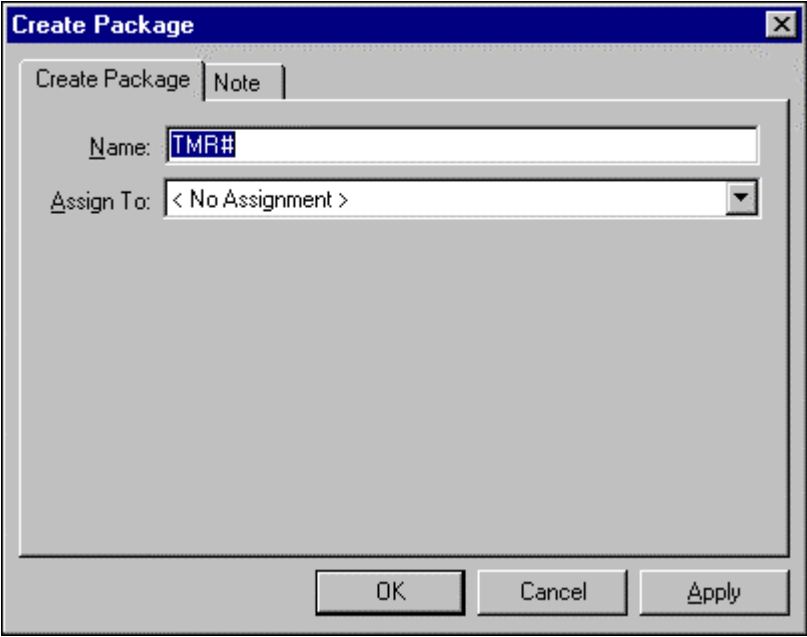
Note: A conflict occurs when a set of lines is modified in both the branch and the trunk; insertions and deletions are not conflicts.

- **Take Trunk Version** automatically selects the trunk (target) to create the final version, without comparing the contents of the versions. Choosing this option creates a normal version on the trunk and closes the branch.

- **Take Branch Version** automatically selects the branch (source) to create the final version, without comparing the contents of the versions. Choosing this option creates a normal version on the trunk and closes the branch.

Create Package

The results of the create package process are dependent on the properties of the process. This process creates a package in the state defined as the initial state in its process properties dialog. In addition, if the form option was selected in the process properties dialog, a form with the same name as the package is created and the two objects are automatically associated. The create package process provides a streamlined method of creating and automatically associating a package and form.



Name—Enter a name in the Name field for the package to be created. If the process was defined to include a form, a form with the same name as the package is automatically created. A default name might have been specified in the process properties dialog; this name should be used as a template to illustrate a naming convention because multiple packages with the same name cannot be created.

Assign To—The Assign To field allows you to assign the package to a user. Use the drop-down list next to the Assigned To field to list all defined Harvest users in your installation. You can select a user name for assignment from this list.

Cross Project Merge

The cross project merge process is used to merge the changes made to items in one project with the changes made to the same items in another project. The merge creates new versions in the target project that are the latest for each item on the trunk. For example, in release-oriented development, some developers can be working on a maintenance release such as Release 1.1, while others are working on major functional changes for Release 2.0. Several items are updated in both projects, and the groups need to combine their changes to update the items in both projects.

Cross project merges are executed on packages. When the cross project merge process is invoked, you select a destination package and one or more source packages for the merge. For the merge to be successful, the source packages cannot contain any reserved or merge-tagged versions. If the placement option is set to trunk only, the destination project cannot contain any reserved or merge-tagged versions of the items being merged; otherwise, a branch version for the incoming items is created.

A removed-tagged (D) item can be carried to the destination view if the item has already been removed in the destination; its new versions cannot be carried from the source. Renamed items in the source are renamed in the destination as well.

All items for which versions are being merged must exist in one or more repositories shared by the two projects. If two projects do not share a repository, a cross project merge between the projects is not possible.

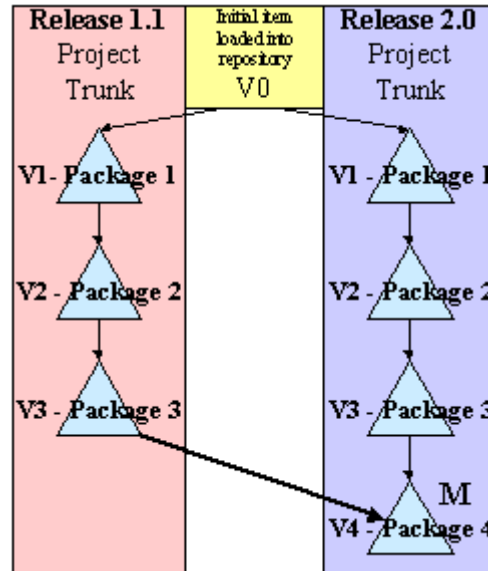
Branch versions are not eligible for a cross project merge. If the source package of a cross project merge contains branch versions, the merge process ignores those versions and executes a normal merge for trunk versions associated with that package. The presence of branch versions does not generate error messages in the merge process.

The versions created in the target project by the cross project merge are located on the trunk, and are the latest on that project's trunk. Each version created as a result of the merge is tagged as merged (M) unless the trunk has not been changed, in which case conflicts do not occur and the source version is simply copied to the trunk of the destination project and not tagged. Merge-tagged versions can be resolved using the interactive merge process.

These restrictions apply to merge-tagged versions of items:

- A merge-tagged version of an item must be resolved through the interactive merge process before another version of the same item can be cross project merged.
- An item that has a merge-tagged version cannot be checked out for Update.
- Packages that contain merge-tagged versions cannot be promoted or demoted to a state in a different view.

In the following example, a package in project Release 1.1 that contains version 3 of an item is cross project merged with a package in project Release 2.0, which contains one version of the item.



The new version 4 created in project Release 2.0 by the cross project merge is the latest on the project's trunk, and is tagged as merged (M). The destination package specified in the cross project merge process is associated with the new version, regardless of whether the package contained a previous version of the item.

Cross project merges can sometimes create versions of an item that did not previously exist in the destination project. For this to occur, the following conditions must be met:

- The source and destination projects must share a repository.
- The item for which versions are being merged must be a new item that was loaded or checked in to the shared repository. If the item was loaded into the repository, the destination project must have been associated with the repository before the item was loaded.
- The item must exist in the source project. This is true:
 - If the new item was checked in from the project.
 - If the project was associated with the repository after the item was loaded.
 - If the item was created in the project by a cross project merge.

When these conditions are met, the cross project merge process creates versions of the item in the destination project. If the source package of the merge contains only version 0 of the item, this version is copied to the destination project as version 0. If the source package contains a non-zero version, the merge creates a version 0, which is the same as the version being merged.

Cross project merges typically produce new, merge-tagged versions in the destination project; however, cross project merges can have different results. The resulting versions depend on the status of the versions in the source package and destination project. The following table shows the results of cross project merges according to the type of version in the source package and destination project.

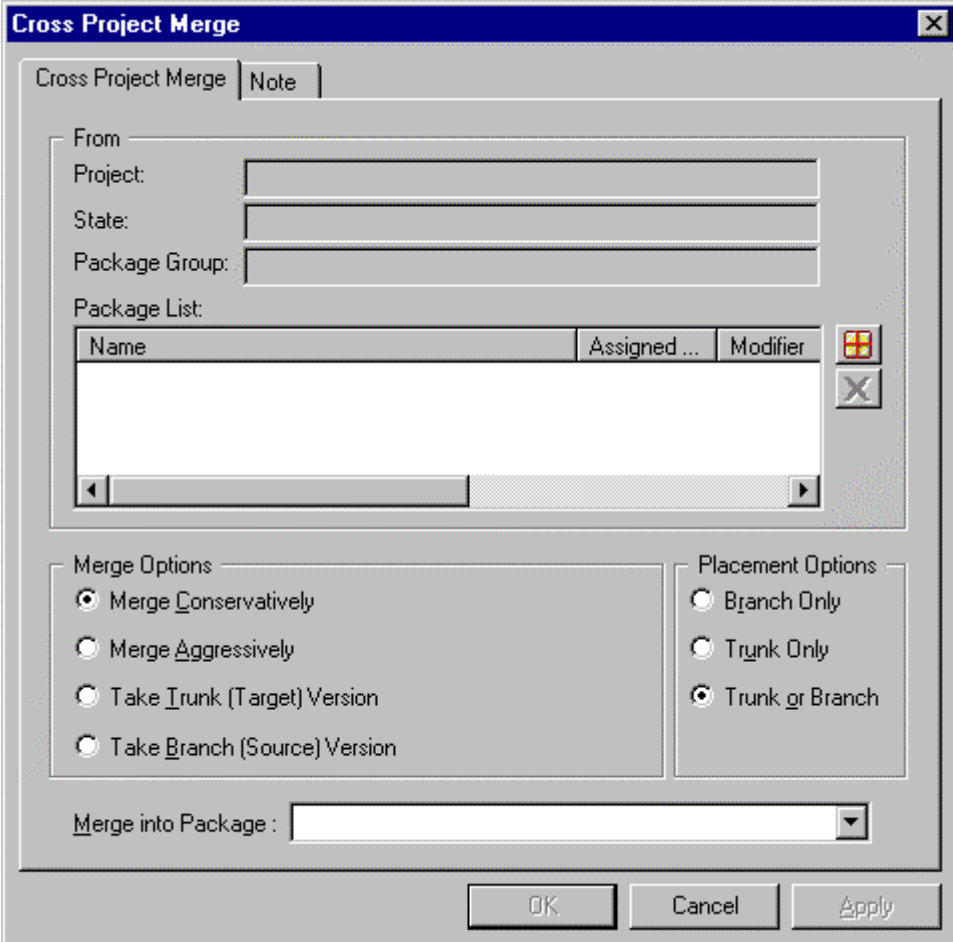
Note: Version 1, as the source or destination version, represents any non-zero version.

Cross Project Merge Results

Source Version	Destination Version	Result in Destination Project
0	0	None
0	1	None
1	0	Version 1 (no tag) created*
1	1 (identical)	Version 2M created*
1	1 (different)	Version 2M created*
1	1D	Version 2M created*
1D	0	Version 1D created*
1D	1	Version 2D created*
1D	1D	None
R	any	Merge fails
1M	any	Merge fails
any	R	Merge fails
any	1M	Merge fails

D=remove tag; R=reserve tag; M=merge tag

* Version number represents the number of the destination version plus one.

The image shows a 'Cross Project Merge' dialog box with a blue title bar and a close button. It has two tabs: 'Cross Project Merge' and 'Note'. The 'Cross Project Merge' tab is active. It contains several input fields: 'From Project:', 'State:', and 'Package Group:'. Below these is a 'Package List' section with a table header containing 'Name', 'Assigned ...', and 'Modifier'. The table body is empty. To the right of the table are two buttons: a red cross icon and a close button. Below the table is a horizontal scrollbar. At the bottom of the dialog are three buttons: 'OK', 'Cancel', and 'Apply'. There are also two sections of radio buttons: 'Merge Options' with 'Merge Conservatively' (selected), 'Merge Aggressively', 'Take Trunk (Target) Version', and 'Take Branch (Source) Version'; and 'Placement Options' with 'Branch Only', 'Trunk Only', and 'Trunk or Branch' (selected). A 'Merge into Package:' dropdown menu is located at the bottom of the main content area.

Project—If you have selected a package on the workbench, the project name to which the package belongs populates this field.

State—If you have selected a package on the workbench, the state name to which the package belongs populates this field.

Package Group—If you have selected a package on the workbench, the package group name to which the package belongs populates this field.

Package List—Clicking the button next to the Package List field opens the Select a Package (cross project) dialog from which you can select the project, state, or package group from which packages are merged (pulled) into your current project, and then select the appropriate packages to merge. When you close the Select a Package (cross project) dialog, the selected packages are displayed in the Package List field, and the Project, State, and Package Group fields are populated with the package's context.

Merge Options—Select **one** of the Merge Options buttons to specify how the process executes. During setup an administrator defined the options available.

- **Merge Conservatively** creates a merge-tagged version, regardless of the contents of the versions. An exception is when the branch version is the latest version in the view; in this case, it is closed and a normal version is created.
- **Merge Aggressively** creates a merge-tagged version only when conflicts are found. If no conflicts are found, the branch and trunk versions are merged to create a normal version.

Note: A conflict occurs when a set of lines is modified in both the branch and the trunk; insertions and deletions are not conflicts.

- **Take Trunk (Target) Version** automatically selects the trunk (target) to create the final version, without comparing the contents of the versions. Choosing this option creates a normal version on the trunk and closes the branch.
- **Take Branch (Source) Version** automatically selects the branch (source) to create the final version, without comparing the contents of the versions. Choosing this option creates a normal version on the trunk and closes the branch.

You can select a rule to automatically specify the target location of the merge version. The Placement Option buttons allow you to specify merge version destinations. Select **one** of the following:

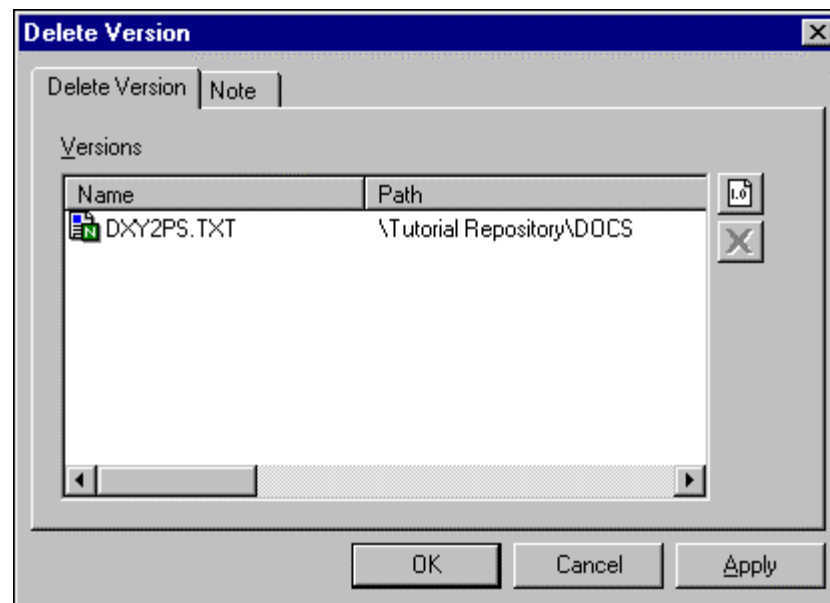
- **Branch Only** creates a merge version on the target branch. This allows changes to be copied from the source project to the target project even if one or more target items are reserved for update in the main trunk. With this option, a branch is created to store the changes. The target package cannot be the same package that contains the items reserved for update on the main trunk.
- **Trunk Only** creates a merge version on the target trunk.
- **Trunk or Branch** creates a merge version on the target trunk or branch. This allows changes to be copied from the source project to the target project even if one or more target items are reserved for update in the main trunk. If items are reserved for update on the trunk, a branch is created to store the changes only if the target package is not the same package that contains the items reserved for update. If items are not reserved for update on the trunk, the items are simply copied to the trunk.

Merge into Package—If you have selected a package on the workbench, the selection is copied into the Merge into Package field. If not, use the drop-down list to select a package in the current state to receive the merged changes.

Delete Version

The Delete Version Process Execution dialog allows you to remove selected versions of items in the Harvest repository. This process has these limitations:

- Only the absolute latest version for an item can be deleted. You cannot delete intermediate versions.
- A version cannot be deleted if it exists in more than one view. This means that if the package that created a version has been promoted, it must be demoted to the state where changes were made before the version can be deleted.
- The initial version of an item (version 0) can be deleted with this process but **only** if the item was initially checked in as a new item. (An initial version created by the load repository function can be deleted through the Repositories tab on the Administrator window.)
- More than one version of an item can be deleted simultaneously as long as the versions are sequential and include the latest version.
- Versions of more than one item can be deleted simultaneously. To delete versions from multiple items, all the versions must have been created by the same package. You can use the package filter in the Find Version dialog to facilitate your selection.



Versions—If you have selected a version on the workbench, it populates the list box. If not, clicking the button next to the list box opens the Select Version dialog from which you can select one or more versions to delete. When you close the Select Version dialog, the selected versions are displayed in the list box. To remove a version from the list, select the version and click the Remove (X) button.

Note: Although you can select versions according to any filtering criteria, all versions to be deleted in one execution must have been created by the same package.

After clicking OK or Apply, a dialog appears in which you can confirm or cancel the delete version process.

Demote

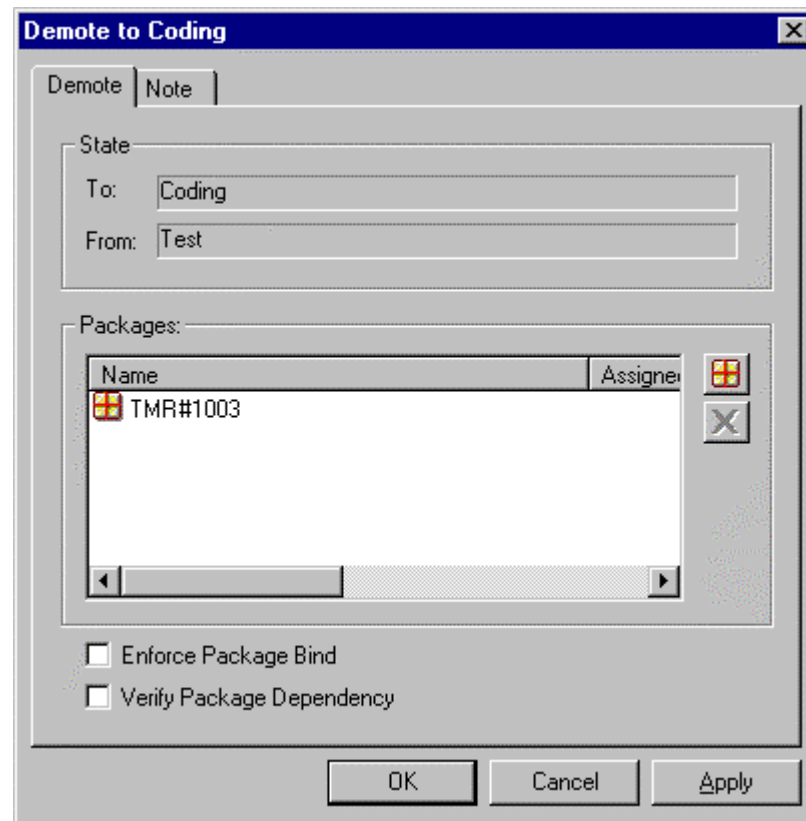
The demote process allows you to send one or more packages in the current state to a previous state in the life cycle. When a package is demoted to a prior state with a different view, all changes introduced into the current view by the package are removed and become visible in the destination view. If approvals are required for promotion, the approval status of the package is reset when it is demoted. If multiple packages are selected for demotion, they are handled as a unit; either all of them are demoted or none.

The possibility of “out of view” versions exists in a life cycle. These are versions that were created in a view, but that currently do not belong to a view. “Out of view” versions can occur when a package, which has been modified in a working view, is demoted to a state that has no view. The demote process removes from the view all of the versions associated with the demoted package. It can also remove new versions created in that view. When the package moves to a state with no view, those removed versions are not visible in any working view. To make the versions visible, the package associated with the versions must be promoted or demoted to a state that has a working view.

The presence of a reserved or merge-tagged version in the package selected for demotion causes the demote process to fail if the demotion includes a change in view. If you attempt to demote a package to a state in a different view, and the package has merged or reserved versions are associated with it, the demote process fails and an error message is generated.

If a package is part of a bound package group, all packages in the group must be demoted together. To demote a package that belongs to a bound package group, you must first unbind the package.

To easily demote packages, select one or more packages, drag them to the state you want, and drop them on the state icon. After successfully completing the drag and drop procedure, the demote dialog opens. The dialog's To and From fields are populated with your current demotion data for confirmation.



To and **From**—The name of the destination state is displayed in the To field and the source state is displayed in the From field.

Packages—If you have selected a package on the workbench, it populates the Packages list box. If not, clicking the button next to the Packages field opens the Select a Package dialog from which you can select one or more packages in the current state to demote. When you close the Select a Package dialog, the selected packages are displayed in the list box. To remove a package from the list, select the package and click the Remove (X) button.

The availability of the options is determined by their setting on the Demote Properties dialog.

- If the option is not enabled on the Demote Properties dialog, then the option is not enabled on the Demote Process Execution dialog. You can decide to enable the option or not when executing the process.
- If the option is enabled on the Demote Properties dialog, then the option is enabled by default on the Demote Process Execution dialog. When the demote process is executed the option is enforced and cannot be overridden by the user.

Enforce Package Bind—Enabling this option causes all the packages belonging to a bound package group to be demoted together.

Verify Package Dependency—Enabling this option disallows packages to be demoted to the destination state until all dependent packages are located in the current state.

Interactive Merge

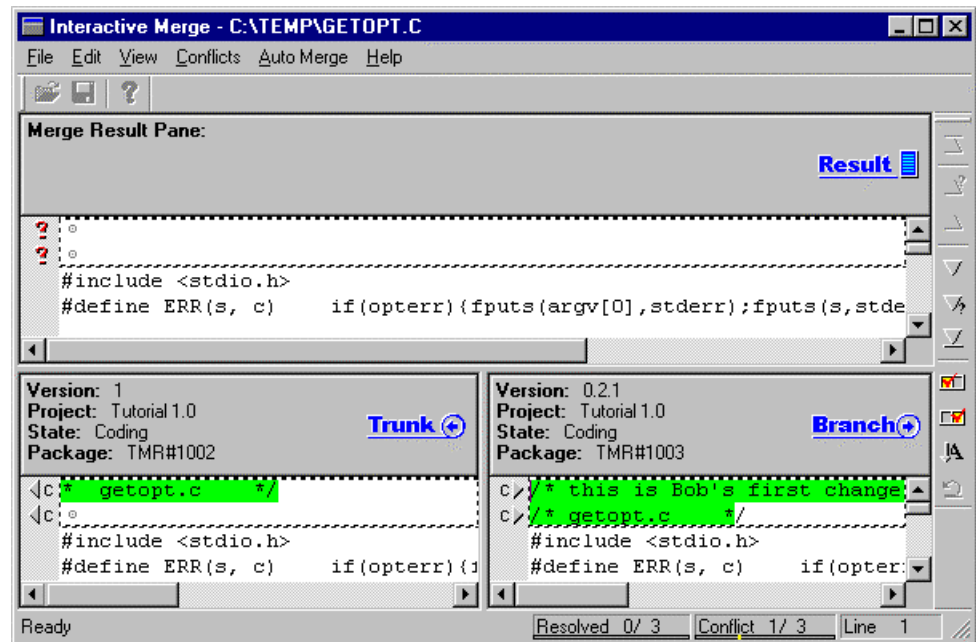
The interactive merge process enables you to combine the changes made on unmerged branches with changes on the trunk or to resolve a merge-tagged version after it is created by the concurrent merge or cross project merge process.

To resolve a merge tag, right-click the merge-tagged version before invoking the interactive merge process from the shortcut menu. When the merge is completed, the merge-tagged version is replaced by a new, untagged version as the latest in the project's trunk.

Binary files cannot be used with the interactive merge process. When this process executes, it first checks that the file consists entirely of printable characters (or control characters such as tab, carriage return, and line feed). If any area of the file contains binary data, the merge process does not execute and an error message is displayed.

The two versions of the item being merged are displayed in the merge execution dialog with common blocks (lines identical in both versions) and conflict blocks. Conflict blocks are positioned side-by-side, and one block must be selected in favor of the other. All blocks are editable enabling you to resolve conflicts.

Note: When the interactive merge process is used to resolve a merge-tagged version created by a cross project merge, the source version of the merge is treated as a branch version in the interactive merge process and the destination version is treated as a trunk version.



The Interactive Merge Process Execution dialog has three panes: the Merge Result pane, the Trunk pane and the Branch pane.

Merge Result Pane—The Merge Result pane is the top pane. The text in the Merge Result pane represents the final version of the file and becomes the new version that is created when the file is checked in.

Trunk Pane—The Trunk pane contains the trunk version of the file being merged.

Branch Pane—The Branch pane contains the branch version of the file being merged.

The panes are synchronized to ensure that all three panes are displaying the same conflict lines.

Text characters beside lines and color codes indicate the status of the text.

- u indicates unchanged text. The color code is black.
- a indicates added text. The color code is yellow.
- c indicates changed (conflict) text. The color code is green.
- d indicates deleted text. Deleted text is indicated by strike marks.
- If the imaginary symbol option is enabled, a circle indicates the lines that have been added to synchronize the panes.

Status Bar—The status bar at the bottom of the dialog contains two meters, Resolved and Conflict. The line number of the current conflict is also displayed in the status bar.

- The Resolved meter shows the number of resolved conflicts and the total number of conflicts. A green horizontal bar indicates the progression of resolved conflicts.
- The Conflict meter shows your current conflict location and the total number of conflicts. A yellow horizontal bar indicates your location in the sequence of conflicts.

Navigation

The Conflicts menu and a toolbar provide you with different options to navigate through the conflicts. The navigation options found in the Conflicts menu are the same as those found on the toolbar. The toolbar can be undocked and relocated on the dialog.

You have the following options:

- **First** moves you to the first conflict.
- **Previous Unresolved** moves you to the previous unresolved conflict.
- **Previous** moves you to the previous conflict.
- **Next** moves you to the next conflict.
- **Next Unresolved** moves you to the next unresolved conflict.
- **Last** moves you to the last conflict.
- **Auto Jump Next** automatically moves you to the next conflict after a conflict has been resolved.

View

The View menu lets you customize the dialog to show or hide the Merge Tools, Toolbar, Status Bar and Imaginary Symbols.

You can double-click the pane's title bar to hide the title bar and increase your visible text space. Double-clicking again on the top border of the pane restores the title bar.

Resolving Conflicts

Options to select the trunk or branch conflicts are available from the Edit menu, shortcut menu, shortcut keys and the toolbar.

- **Take Trunk** chooses the conflict on the trunk.
- **Take Branch** chooses the conflict on the branch.

Two options allow you to reset conflicts to an unresolved condition.

- **Reset Current Conflict** restores the current conflict block to its original condition. This option is available from the toolbar and the Edit menu.
- **Reset All Conflicts** restores the all conflict blocks to their original condition. This option is available from the Edit menu.

Cut <Ctrl+X>, copy <Ctrl+C>, and paste <Ctrl+V> functions are active in the interactive merge dialog and can be used in any text block. All blocks are editable and any changes can be made to resolve conflicts.

Auto Merge Options—From the drop-down list, you can select **one** of the Auto Merge options to specify how the process executes. During setup the administrator defined the options available.

Note: A *conflict* occurs when the same line or block of data is modified in both the branch and the trunk. Insertions and deletions are considered *changes*.

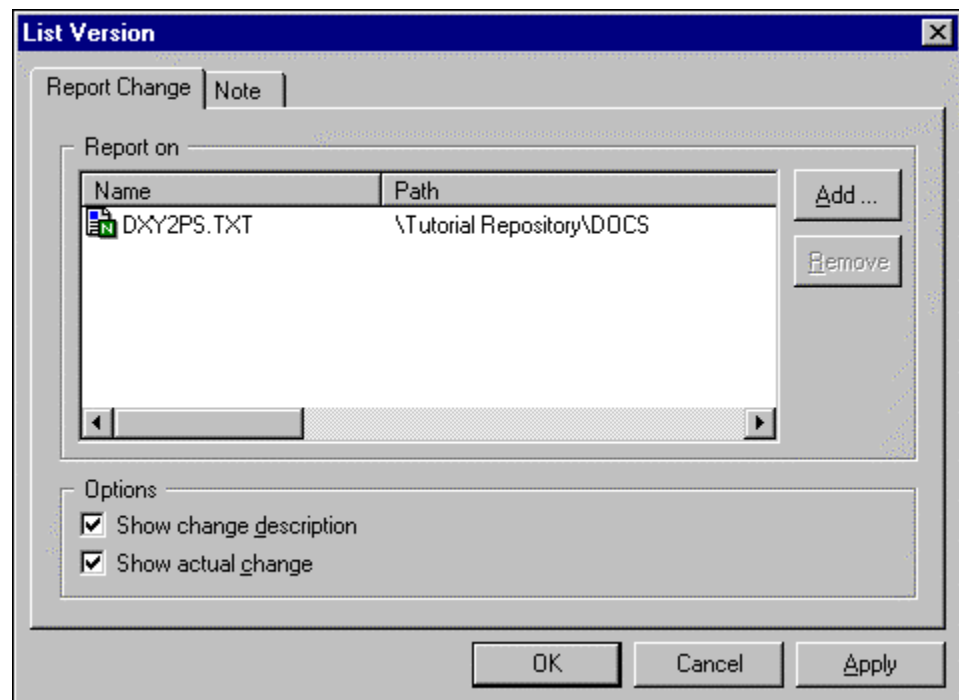
- **Merge All Changes Except Conflicts** automatically merges the changes but does not include conflicts. You must then resolve the conflicts. Choosing this option creates a normal version on the trunk and closes the branch.
- **Merge All Changes, Take Trunk Conflicts** automatically selects all changes and selects the branch conflict when a conflict between the trunk and branch occurs.
- **Merge All Changes, Take Branch Conflicts** automatically selects all changes and selects the trunk conflict when a conflict between the trunk and branch occurs.
- **Take All Trunk Changes** automatically selects the trunk (target) to create the final version, without comparing the contents of the versions. Choosing this option creates a normal version on the trunk and closes the branch.
- **Take All Branch Changes** automatically selects the branch (source) to create the final version, without comparing the contents of the versions. Choosing this option creates a normal version on the trunk and closes the branch.

Two execution options are available on the dialog and in the File menu: Check In and Close.

- You cannot execute the interactive merge until all conflicts are resolved. The Check In option is disabled until all conflicts are resolved. Similarly, the Check In button on the dialog does not display until all conflicts are resolved. When all conflicts are resolved, click Check In. Checking the file in replaces the merge-tagged version with a new, untagged version.
- If you do not want to merge the files, click Close. A dialog opens in which you can confirm or cancel the Close. Close disregards all your selections, and the file being merged is returned to its original unmerged state.

List Version

The list version process allows you to generate reports about the changes made to items in the current project. This process is useful for viewing changes made to an item to create new versions on the trunk. The listing produced by this report is in the same format as that produced by the UNIX diff command. The List Version report is automatically written to the output log, from which it can be copied to the clipboard or saved to a text file.



Report on—If you have selected versions on the workbench before invoking this process, they are displayed in the list box. You can make a selection directly from this dialog by clicking the Add button to open the Find Version dialog. To remove a version from the list, select the version and click the Remove button.

Two check boxes let you select options for this report:

- **Show change description** causes the change description provided during check in, to be displayed.
- **Show actual change** causes the actual line-by-line changes between one version and the next to be displayed.

The differences are displayed as instructions that can be used to change the first version and make it the same as the second. These instructions contain either: add a, delete d, or change c commands. A line number follows each command. A less than sign (<) precedes lines from the first version. A greater than symbol (>) precedes lines from the second version. A pair of line numbers separated by a comma represents a range of lines; a single line number is used to represent a single line.

The differencing algorithm converts the first version into the second. The line numbers to the left of each of the a, c, or d instructions always pertain to version 1; numbers to the right of the instructions apply to version 2. To convert version 1 to version 2, you can ignore the numbers on the right.

Examples

The following examples are based on versions of a shopping list. You can apply this algorithm to any kind of code. The original example list looks like this:

```
list1
tomatoes
potatoes
corn
```

Example 1: Deleting a Line

You then remove potatoes from the list and it looks like this:

```
list2
tomatoes
corn
```

The List Version report would display the following output when list1 is compared to list2:

```
2d1
< potatoes
```

This output indicates that you must delete line 2 to make the first list like the second. The line to be deleted is displayed below the command. The less than sign (<) indicates that this line is from list1.

Example 2: Adding a Line

Then instead of leaving out potatoes, you add spinach to the list. The second list now looks like this:

```
list2
tomatoes
potatoes
spinach
corn
```

The output of differencing this list against list1 would look like this:

```
2a3
> spinach
```

This output indicates that you need to add a line to list1 after line 2. The line to be added is from the second list as indicated by the greater than sign (>) and is listed below the instruction.

Example 3: Changing a Line

In the third example, you buy peas instead of potatoes. The revised shopping list now looks like this:

```
list2
tomatoes
peas
corn
```

The output of differencing this list against list1 would look like this:

```
2c2
< potatoes
---
> peas
```

This output indicates that line 2 needs to be changed. The line from the first file (< potatoes) needs to be changed to be like the line from the second file (> peas). The three hyphens indicate the end of the text in the first list and the start of the text in the second list that should replace it.

Example 4: Changing Several Lines

In the fourth example, you throw out the old list and replace it with fruit:

```
list2
apples
pears
bananas
```

The output of differencing this list against list1 would look like this:

```
1,3c1,3
< tomatoes
< potatoes
< corn
---
> apples
> pears
> bananas
```

This output indicates that a range of lines (1-3) in the first list must be changed and replaced with a similar range of lines from the second.

Move Package

The move package process moves one or more packages from the current project and state to a state in another project. This process moves only the package definition and history, not any changes made in the first project. Changes associated with packages cannot be moved with this process.

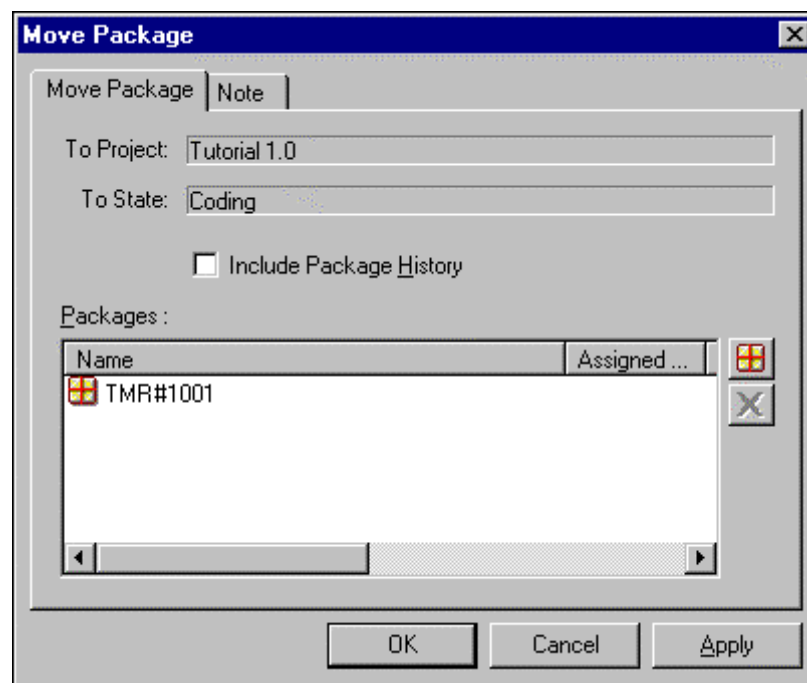
Moving packages is typically used to link a problem tracking project to a change control project. For example, a customer support group records problems using Harvest forms. Packages are created in the support project and associated with various problem forms. These packages can move through different states in the support project until they are ready to be assigned. Rather than creating a new package in the development project, the original package with its history intact and any form associations can be moved from the support project to the development project. If multiple packages are selected, they are all moved as a unit.

In order for the package process to succeed, Harvest verifies that all packages selected meet the following criteria:

- No versions associated with the package can exist in the source project.
- A package with the same name cannot exist in the target project.
- If an approval process is in place for a state, a package cannot be moved until it has been approved.
- The user executing the process must have access to create packages in the destination project.

The following list indicates what happens to various package attributes in the new project:

- The package note is retained intact.
- All form associations are maintained.
- The package groups are not carried across to the new project because package groups are not global objects. You must edit the package in the new project to add it to any groups.
- The package history is moved with the package if the appropriate option is selected, and an additional record is added to the history indicating when it was moved.



To Project—The default destination project is displayed.

To State—The default destination state is displayed.

Include Package History—You can include the package history by checking the Include Package History option. This transfers all the history records that have been created in former projects. If the history is no longer required, excluding it can reduce processing time.

Packages—If you have selected a package on the workbench, the selection is copied into this field. If not, clicking the button next to the Package field opens the Select a Package dialog. From the Select a Package dialog, you can select one or more packages in the current state to move. When you close the Select a Package dialog, the selected packages are displayed in the list box.

Notify

The notify process allows you to send a mail message to individual users or all members of one or more user groups. When a notify process is set up, a default message can be specified, as well as default users and groups to notify. These defaults determine how the execution dialog appears when it is first accessed. Users can override the defaults.

The mail program to use and the destination of any program output are also specified during setup, but these aspects of the notify process cannot be modified at execution time. On UNIX-based clients, the mail program executes on the client machine. On PC clients, the mail program is executed from the server machine.

When mail is executed on the server, the message header generated by the mail program is affected. It records that the user who started the server process sent the message. Harvest adds a line following the header generated by the mail program, indicating the name of the client user who initiated the notification process.

Notify

Notify | Note

Name: Notify

Mail Utility:

Subject: Harvest Message

Mail Message:
Packages promoted:
[packages]

Display
Output: Display Errors: Display

Users to Notify:

Name	Real Name

Add Delete

User Groups to Notify:

Name

Add Delete

Creation Date: 5/29/2001 10:24:49 Creator: harvest
Modified Date: 5/29/2001 10:24:49 Modifier: harvest

OK Cancel Apply

Name—The name of the notify process is displayed.

Mail Utility—In the Mail Utility field, enter the name of a mail program and any arguments you want to supply when the program is invoked. Specify a full path to the program.

- On UNIX clients, Harvest searches for a path in the PATH variable of the user executing the process.
- On PC clients, the mail program is executed on the server machine and must be in the path of the user who started the server. To navigate to and select a mail utility, click the browser button next to this field to open the operating system file dialog.

Subject—A default subject might have been specified in the process properties dialog and is displayed. The subject appears on the subject line of the email message when it is sent. You can modify the content of this field.

Mail Message—In the Mail Message field, enter a message to be sent by this notify process. This field allows you to use up to 2,000 characters.

For example, if this process is going to be used to notify a manager that a package has been promoted, you can specify the appropriate message in this field. If the notify process is being linked to another process, you can use system variables in the messages to represent various parameters.

When the package or version system variable is used in this field, you can right-click packages on the workbench, choose a notify process from the shortcut menu and the package or version names are listed in the notify message.

Display—You can select a default action to be taken with any general or error outputs generated by the notify process by selecting an option in the Output and Errors menus. The choices are Display or Discard. All output is automatically appended to the output log if the Display choice is selected.

Users to Notify—The Users to Notify list allows you to specify the users who receive the notification. To add a user to the list, click the Add button, and then select the user and click OK. To delete a user from the list, select the user and click the Delete button.

User Groups to Notify—The User Groups to Notify list allows you to specify the user groups who are notified. To be notified, users in the group must have use access to the project unless they are listed in the Users to Notify list. To add a group to the list, click the Add button, and then select the group and click OK. To delete a group from the list, select the group and click the Delete button.

Promote

The promote process allows you to send one or more packages in the current state to the next state in the life cycle. If an approval process exists for a state, approvals are verified before a package can be promoted. When a package is promoted to a state with another view, all its changes located on the trunk become visible in that view.

These conditions affect whether a package can be promoted:

- If an item is currently reserved by the package, the package can only be promoted if the next state is in the same view. If the promotion includes a change in view, all items in the package must be checked in (unreserved).

Note: For the purpose of this restriction, promoting from a state with a view to a state without a view is considered changing views.

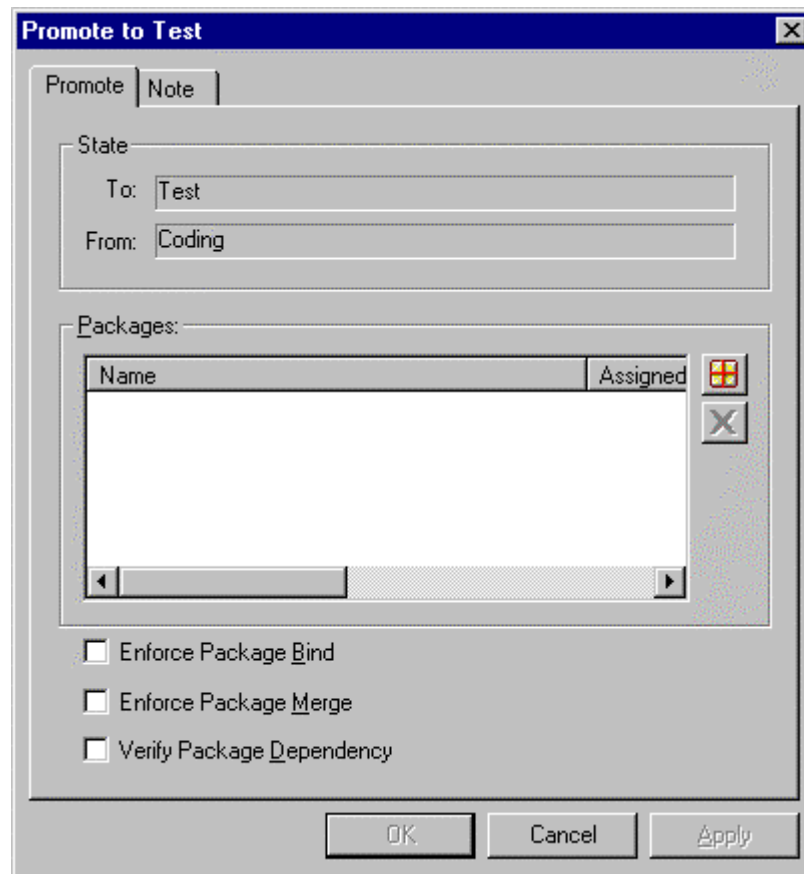
- If one or more approval processes exist in a state, a package cannot be promoted out of that state until all approvals have been given for at least one approve process.
- If a user has rejected a package, the package cannot be promoted until that user executes the approval process.

If multiple packages are selected for promotion, they are promoted as a unit.

Packages must meet the following criteria for their promotion to succeed:

- If the process does not allow unmerged packages to be promoted, the latest change for a package must not be on a branch.
- If the promotion includes a change in view, no items can be reserved by the package.
- If a state has an approval process, all approvals must have been given.
- If a package is part of a bound package group, all packages in the group must be promoted together. To promote a package that belongs to a bound package group, you must first unbind the package.

To easily promote packages, select one or more packages, drag them to the state you want, and drop them on the state icon. After successfully completing the drag and drop procedure, the promote dialog opens. The dialog's To and From fields are populated with your current promotion data for confirmation.



State To and From- The State To and From fields display the name of the destination state and source states.

Packages—If you have selected a packages on the workbench, they populate the Packages field. If a package belongs to a bound package group, all packages belonging to the group are listed in this field.

Clicking the Add button next to the Packages field opens the Select a Package dialog, from which you can select one or more packages in the current state to promote. When you close the Select a Package dialog, the selected packages are displayed in the list box. To remove a package from the list, select the package and click the Remove button.

The availability of the options is determined by their setting on the Promote Properties dialog.

- If the option is not enabled on the Promote Properties dialog, then the option is not enabled on the Promote Process Execution dialog. You can decide to enable the option or not when executing the process.
- If the option is enabled on the Promote Properties dialog, then the option is enabled by default on the Promote Process Execution dialog. When the promote process is executed the option is enforced and cannot be overridden by the user.

Enforce Package Bind—Enabling this option causes all the packages belonging to a bound package group to be promoted together.

Enforce Package Merge—Enabling this option prohibits packages to be promoted to the next state if they are associated with branch versions. If you enforce that packages must be merged, the latest change for the package must be on the trunk, not on a branch.

Verify Package Dependency—Enabling this option disallows packages to be promoted to the destination state until all dependent packages are located in the current state.

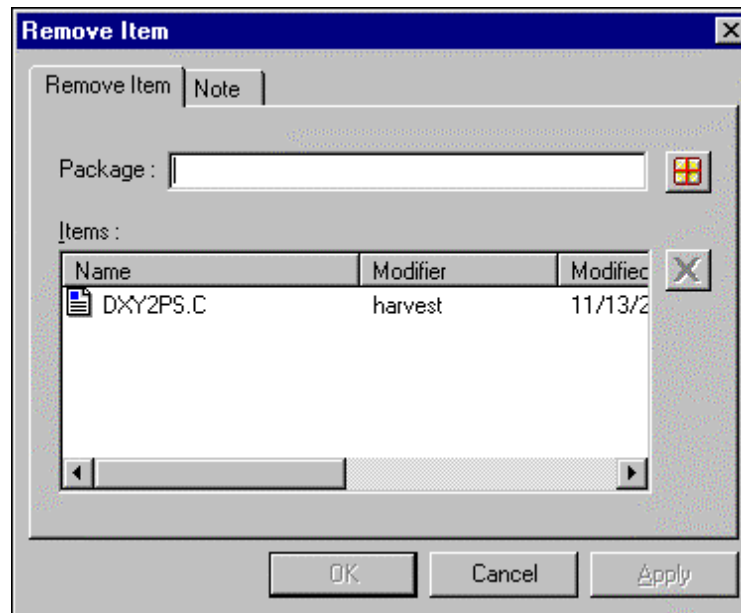
Remove Item

The remove item process allows you to logically delete selected items from a view. When an item has been removed from a view, the item is not really gone; the remove item process creates a new version tagged as removed (D). This version has attributes like other versions and can be seen through the Find Version dialog or in version view on the workbench but not in item view.

Because this process actually creates a version, a comment can be supplied during execution and is associated with the removed version. This comment can be displayed with the List Version report (see the section List Version in this chapter for details).

To restore an item that has been removed, the delete version process can be used to delete the remove-tagged version.

The items must be selected before the process is invoked.



Package—Clicking the button next to the Package field opens the Select a Package dialog from which you can select a package, in the current state, that is to be associated with the removal of the item. When you close the Select a Package dialog, the selected package populates the Package field.

Items—The items you selected on the workbench are displayed in the Items field. To remove items from your selection, select items in the list box and then click the Remove (X) button.

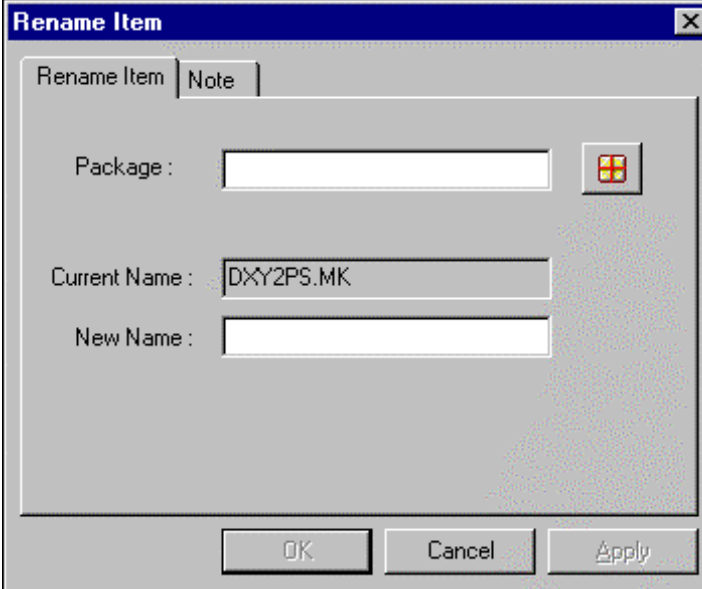
Rename Item

The rename item process allows you to rename an item. The item must be selected on the workbench before the process is invoked. After executing this process, the renamed item is a new version. Previous versions of the item exist with their former names and can be seen in the version view of the item. To see all the versions associated with a newly renamed item, double-click the item name; this lists all versions associated with that item. To restore an item that has been renamed, use the delete version process.

Important! An item can only be renamed if no unmerged branch versions of the item exist. If unmerged branch versions do exist, you must merge the versions to the trunk before renaming the item. If the latest version of an item has an M-tag, interactive merge must be completed on that item before it can be renamed. If the latest version is reserved, the R-tag must first be resolved by checking in the item before renaming it.

Items that have been removed using the Remove Item process (D-tagged) cannot be renamed because they are no longer listed in the item list.

If the repository is shared by more than one project, an item can be renamed only if it is given a unique name within each project.

A screenshot of a 'Rename Item' dialog box. The dialog has a title bar with 'Rename Item' and a close button. It contains two tabs: 'Rename Item' (selected) and 'Note'. The 'Rename Item' tab has three text input fields: 'Package :', 'Current Name :', and 'New Name :'. The 'Current Name' field contains the text 'DXY2PS.MK'. To the right of the 'Package' field is a small icon of a red square with a yellow cross. At the bottom of the dialog are three buttons: 'OK', 'Cancel', and 'Apply'.

Package—Clicking the button next to the Package field opens the Select a Package dialog from which you can select a package, in the current state, that is to be associated with the renamed item. When you close the Select a Package dialog, the selected package populates the Package field.

Current Name—The name of the item you selected on the workbench is displayed in this field.

New Name—Enter a new name for the item.

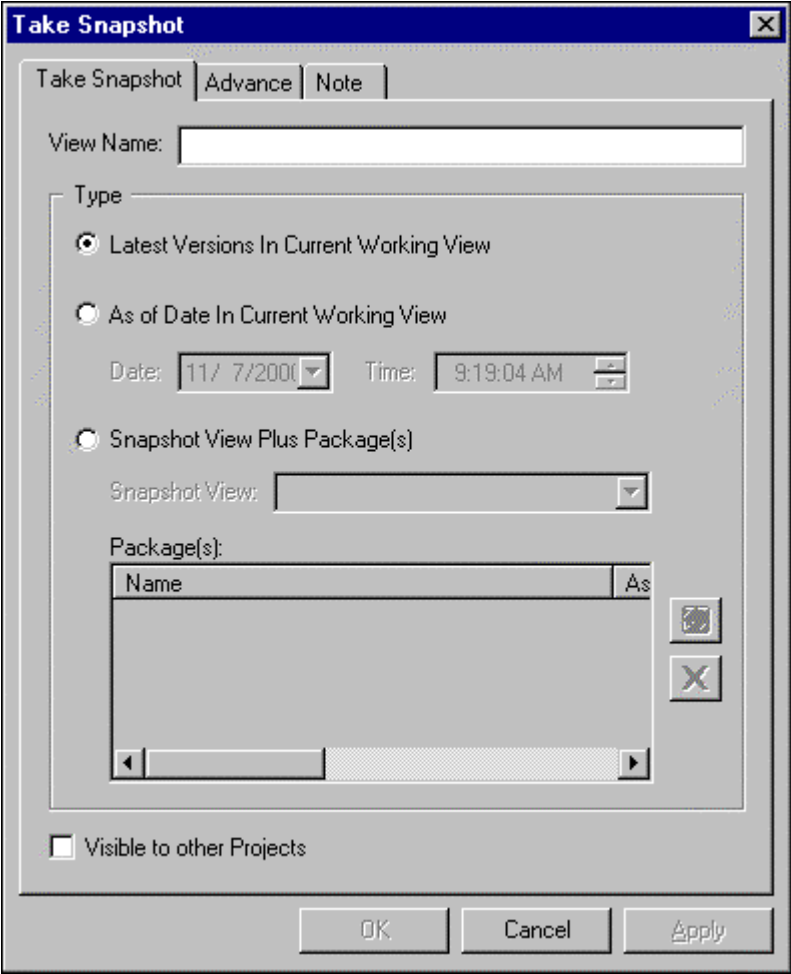
Take Snapshot

The take snapshot process creates a snapshot view of the current working view. Snapshot views are read-only images of working views at a certain point in time that allow administrators to capture a software inventory at significant points in its development. Once a snapshot has been created, it can be used to support other application management functions, such as baselining or re-creating an application at a certain point in time.

These rules apply to the take snapshot process:

- If the current working view contains reserved versions, the take snapshot process uses the untagged trunk versions. In order to use the reserved versions, the reserved versions must be checked in before executing the process.
- If empty item paths exist in the paths you select, they are included in the snapshot view.
- A take snapshot process fails if the current working view contains merge-tagged versions. Merged versions must be resolved before executing the process.

Snapshot views can also be created using the Snapshot View Properties dialog. The take snapshot process, however, provides a streamlined and easily accessible method for creating snapshots.



The "Take Snapshot" dialog box is shown with three tabs: "Take Snapshot", "Advance", and "Note". The "Take Snapshot" tab is active. It contains a "View Name:" text field. Below it is a "Type" section with three radio button options: "Latest Versions In Current Working View" (selected), "As of Date In Current Working View", and "Snapshot View Plus Package(s)". The "As of Date In Current Working View" option has a "Date:" field set to "11/ 7/2001" and a "Time:" field set to "9:19:04 AM". The "Snapshot View Plus Package(s)" option has a "Snapshot View:" dropdown menu. Below this is a "Package(s):" section with a table header "Name" and "As". The table is empty. To the right of the table are two buttons: a square button with a plus sign and a square button with an "X". At the bottom of the dialog is a checkbox labeled "Visible to other Projects" and three buttons: "OK", "Cancel", and "Apply".

View Name—In the Name field, enter a name for the snapshot to be created. A default name might have been specified in the process properties dialog. The name should be used as a template to illustrate a naming convention because multiple snapshots with the same name cannot be created.

Latest Versions in Current Working View—By enabling this button, the latest versions in the current working view is captured by the take snapshot process. If the latest version in the working view has been reserved, the latest one with a normal tag in the trunk is selected.

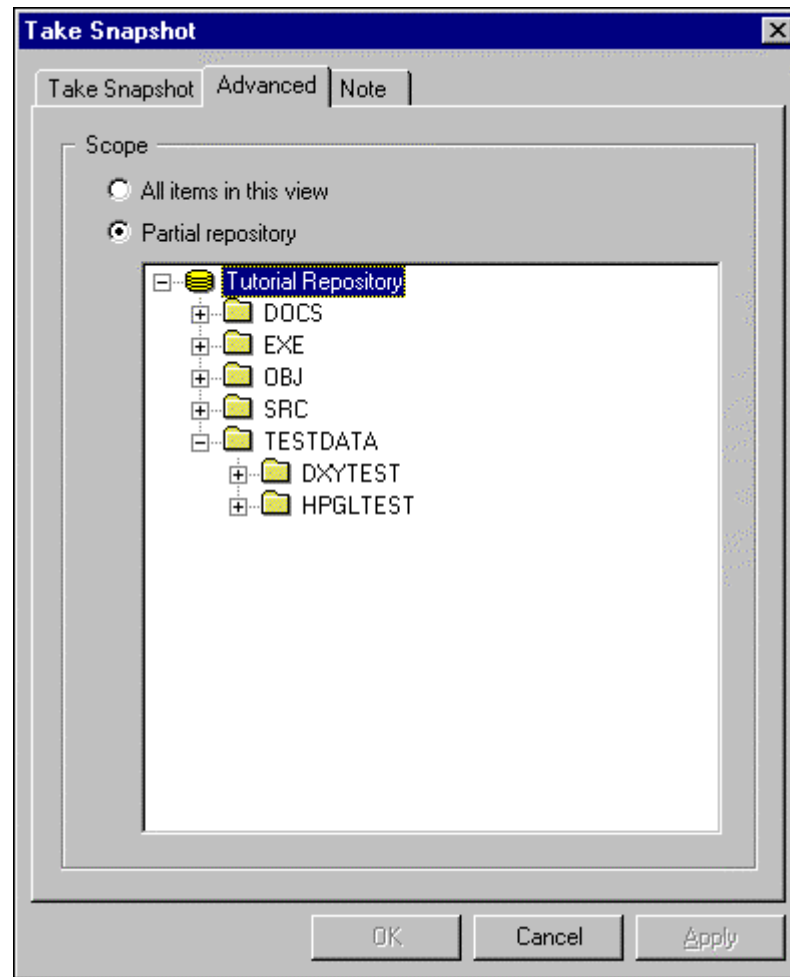
As of Date in Current Working View—By enabling this button, you can specify an earlier date and time by using the drop-down lists or accept the default of the current date and time. All the versions existing in the current working view at the specified date and time is captured by the take snapshot process.

Snapshot View Plus Package—By enabling this option, the versions contained in the snapshot view specified in the Snapshot View field plus the packages listed in the Packages list box is included in the new snapshot.

Snapshot View—Select a snapshot view from the current project to include its versions in the new snapshot.

Package—Initially this list box is empty. You can select packages from the current state to include in the snapshot by clicking the button next to this field to open the Select a Package dialog; the packages you select populates the Packages field. You can remove packages by selecting them and clicking the remove (X) button.

Visible to other Projects—Working views are part of a project and are only accessible from within it. Snapshot views, however, can be marked so that they can be used in the baseline view of other projects. Enable the Visible to Other Projects check box if you want this snapshot to be listed on the Configure Baseline dialog during the setup of the baseline view of other projects. Typically, only snapshots that represent significant phases of development should be made externally visible.



The Advanced tab enables you to select all items in a view or to select paths in a repository.

All items in this view—By choosing this option, the snapshot captures all items in the current working view.

Partial repository—Using this option, you can navigate a repository to select paths to include in the snapshot.

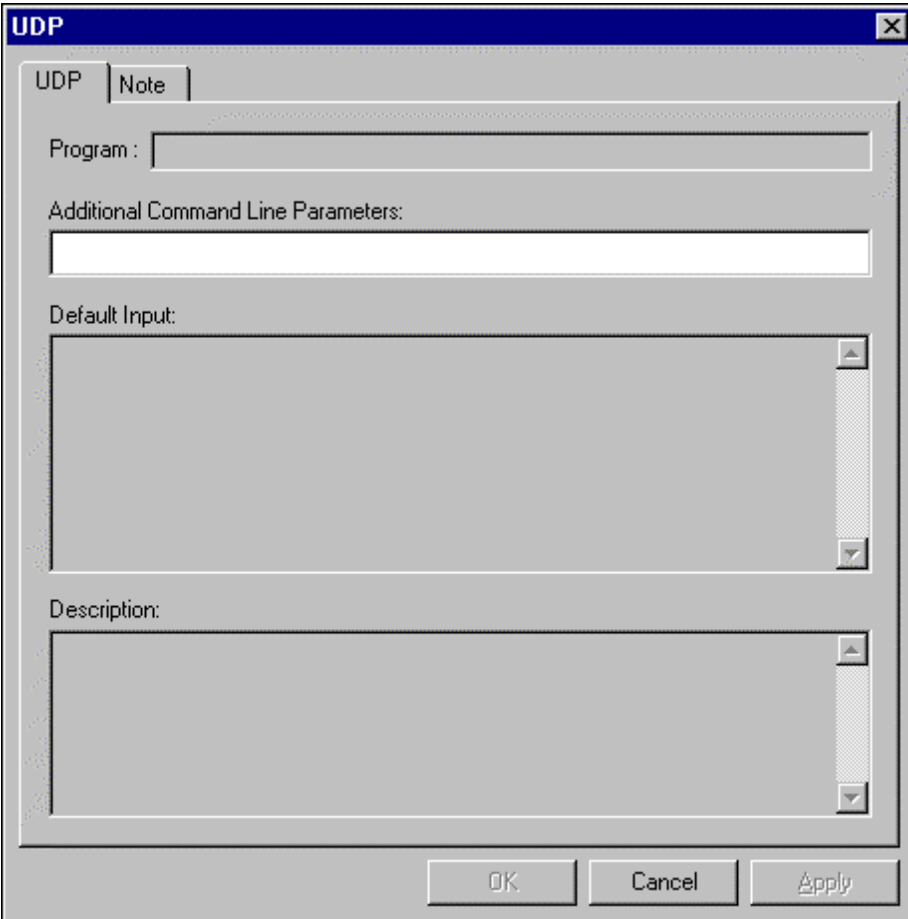
User-Defined Process

The user-defined process (UDP) allows you to invoke an external program to run as a process within your life cycle. The program to execute, any command line parameters, and the options for its output were specified during definition of this process and cannot be modified during execution.

If you select packages or versions on the workbench list view, right click and choose a UDP or notify process from the shortcut menu, the [package] or [version] system variables are expanded to a list.

For programs that read from the default input device, default input parameters might have been specified. These can be overridden or modified at execution time if this field is defined as modifiable in the process properties dialog. You can also supply additional command line parameters.

Note: Processes invoked from Harvest are executed in synchronous mode. Once the execution of a user-defined process begins, you are unable to initiate any other action from the keyboard until it completes.

The image shows a Windows-style dialog box titled "UDP" with a close button (X) in the top right corner. Inside the dialog, there are two tabs: "UDP" and "Note", with "UDP" currently selected. Below the tabs, there are four main sections: 1. "Program:" followed by a single-line text input field. 2. "Additional Command Line Parameters:" followed by a multi-line text input field. 3. "Default Input:" followed by a large, empty multi-line text area with vertical scrollbars. 4. "Description:" followed by another large, empty multi-line text area with vertical scrollbars. At the bottom of the dialog, there are three buttons: "OK", "Cancel", and "Apply".

Program—The Program field is a read-only field that displays the name of the program being executed and any command line parameters defined during setup.

Additional Command Line Parameters—The Additional Command line Parameters field allows you to enter any additional command line parameters to the program executed by this UDP. This allows you to tailor the command invocation.

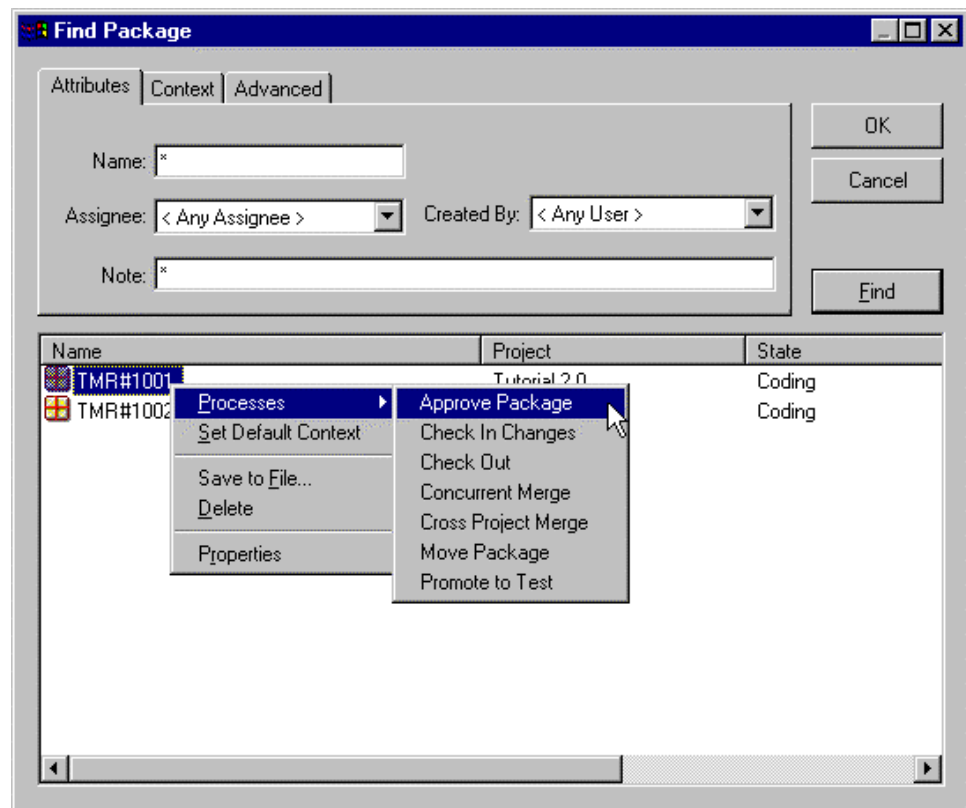
Default Input—Default input parameters to the program executed by this process might have been provided during setup. If the default input was defined as modifiable in the process properties dialog, you can override these parameters by editing them in the Input scroll list. This field only applies to programs that read from the standard input device. Up to 2,000 characters of input can be supplied.

Description—You can type a description of the UDP.

The Find Utility

This chapter describes in detail each Harvest dialog that enables you to locate objects. These dialogs display a list of available objects of a certain type. You can use the dialog to simply browse a list or to select a value to populate a field in the parent dialog.

The Find dialogs, such as the Find Package dialog, enable you to locate objects by specifying filtering criteria and to execute available processes on your selection. For example, after locating all packages awaiting your approval, you could execute the approve process by right-clicking each package and choosing the approve process from the shortcut menu to invoke the Approve process execution dialog.



The buttons on the dialogs vary, depending on if it is a Find dialog or a Select dialog.

- **Cancel** closes the dialog. If you have made any selections in the list, they are not returned to the parent dialog from which the dialog was invoked.
- **Close** closes the dialog.
- **Find** initiates a search and returns the results to the list.
- **OK** returns the selected object or objects to the parent dialog from which the dialog was invoked. Double-clicking an object performs the same function as selecting the object and clicking OK.
- **Stop** halts a search.

To make a selection from the dialog to return to the calling dialog:

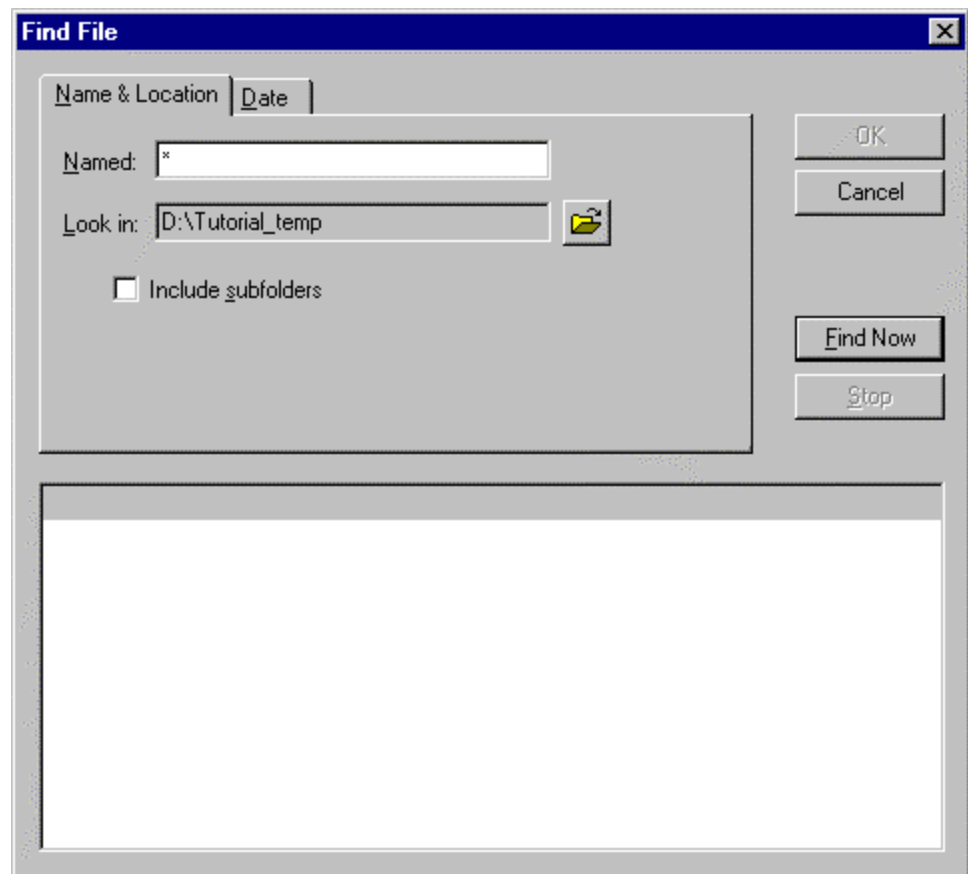
1. Double-click the object name
 - or
 - Select the name and click the OK button.

Find File

The Find File dialog allows you to locate files and to select files from a list. It also supports multiple filtering operations to let you search precisely. In addition, you can use the Find File dialog simply to obtain file information, which can be output to the log.

Note: The Find File dialog can be opened from a number of process execution dialogs; it is not an option on the Tools menu.

When the Find File dialog is initially displayed, its list is empty. You must click the Find button for the current filtering options to take effect. Because the previous filtering criteria might not be relevant to the operation you now want to perform, this implementation prevents unnecessary processing time when the dialog is invoked. Once Find is clicked, the files matching the filtering criteria are displayed in the list and are selected. To return all files to the calling dialog, click the OK button.



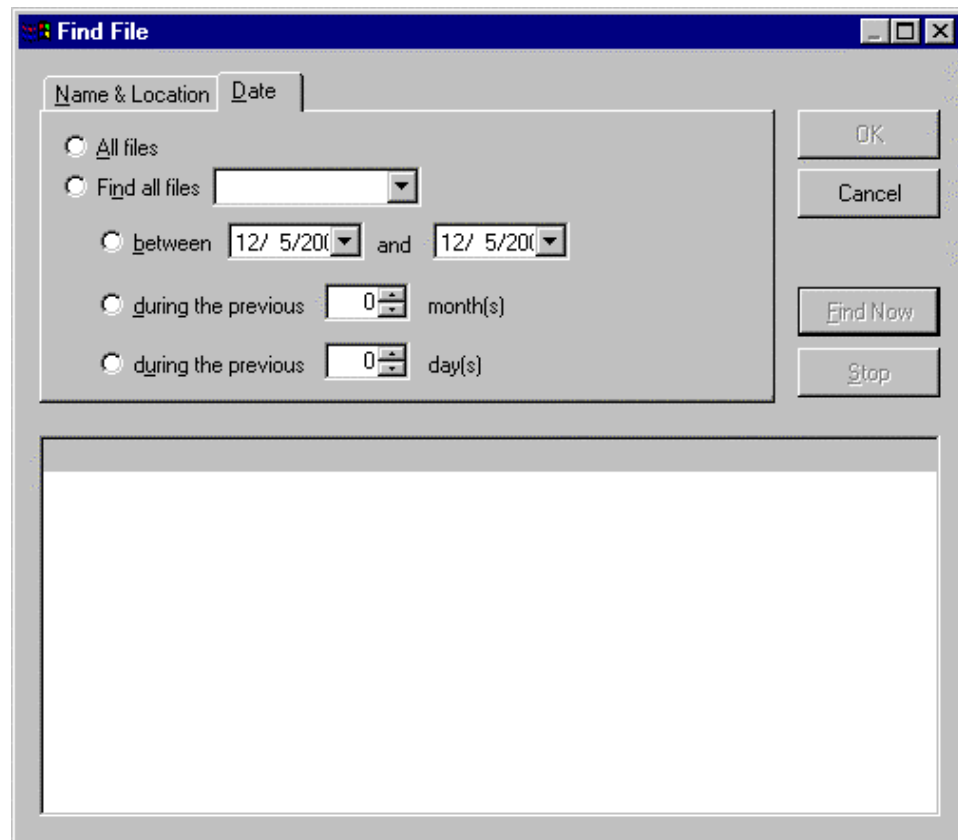
Named—The Named field allows you to filter files according to a naming pattern. You can use any number of wild cards (*) in any position for multiple character matching. The use of the question mark (?) is also supported for single character matching.

You can specify multiple file types at the same time. For example, you can enter: *.cpp, *.h, *.txt, to locate cpp, h and txt files.

Look in—The Look In field allows you to specify a directory to search for a file. Clicking the button next to this field opens the Select a Directory Path dialog, which lists directories in the current client directory. You can traverse through the file system by double-clicking a directory or you can choose a directory by selecting it and clicking the OK button.

Include subfolders—Checking the Include subfolders option displays files in every directory below the current one that match the file name filtering criteria.

When you select Include subfolders and click the Find button, Harvest expands each directory so that the file names displayed in the list include relative paths from your current position.



The Date tab allows you to specify a date range for the search. By default, the All files button is selected, indicating date checking will not be performed. Selecting the Find all files button enables you to select different date searches that can be performed. You can select to search between two dates, search the previous x number of months, or search the previous x number of days. Checking one of these searches enables the Find all files button.

Find Form

The Find Form dialog allows you to execute complex filtering operations to locate forms with common attributes. The Find Form dialog can also be used to select forms according to their association with packages.

A right-click a form in your **form** search results in the list box, to open a shortcut menu with View Form and Save List to File options.

- Choosing View Form opens the form in the form viewer, allowing you to view or modify the form.
- Choosing Save List to File opens a dialog that enables you to save your current search results to a specified name and location.

A right-click a package in your **package** search results in the list box, to open a shortcut menu with available Processes, View Form, Save List to File, Delete and Properties options.

- Choosing View Form lists the associated forms. You can choose a form from the list to open the form in the form viewer, allowing you to view or modify the form.
- Choosing Save List to File opens a dialog that enables you to save your current search results to a specified name and location.
- Choosing Properties opens the Package Properties dialog for the selected package.

Forms exist at the Harvest level. This means that they are available in all projects defined within a Harvest installation and they can be accessed without any context. You can invoke the Find Form dialog by choosing Tools, Find Form from the workbench or by clicking the Find Form button on the toolbar.

Whereas forms exist at the Harvest level, associated packages are located in a state and project. In these contexts, the associated packages can be used to determine which forms to display:

- If your context is at the project level (no state specified), only forms associated with packages in this project are displayed.
- If your context is at the state level, only forms associated with packages in this project currently located in this state are displayed.

The screenshot shows a 'Find Form' dialog box. It has a title bar with the text 'Find Form' and a close button. Below the title bar are two tabs: 'Attributes' and 'Form'. The 'Form' tab is selected. The dialog contains several input fields and buttons. The 'Name' field has a text box with an asterisk (*). The 'Project' field has a dropdown menu with 'Tutorial 2.0' selected. The 'State' field has a dropdown menu with 'Coding' selected. The 'Creator' field has a dropdown menu with '< Any User >' selected. Below these fields is a section labeled 'Show list of:' with two radio buttons: 'Forms' (which is selected) and 'Associated Packages'. To the right of the input fields are three buttons: 'Close', 'Find', and 'Stop'. At the bottom of the dialog is a large, empty rectangular area, likely for displaying a list of search results.

Name—The Name field allows you to filter forms according to a naming pattern. If you leave the default wild card (*), forms are displayed regardless of name. You can use any number of wild cards (*) in any position for multiple character matching. The use of the question mark (?) is also supported for single character matching. The Name field is not case-sensitive, for example, entering N* displays forms whose names begin with N or n.

Project—To search for forms located in only one project, use the drop-down list to select a project for filtering. Using the wild card (*) searches all projects for forms matching your other filtering criteria.

State—If a project has been selected, you can filter for forms located within a state of that project. Use the drop-down list to select a state for filtering. Using the wild card (*) searches all states for forms matching your other filtering criteria.

Creator—This field allows you to filter forms based on the name of the user who created them. Use the drop-down list to select a user for filtering from a list of all Harvest users. Using the wild card (*) searches all users for forms matching your other filtering criteria

Show List of Forms or Associated Packages—Select **one** of the buttons to show a list of forms or associated packages that match your search criteria. After your initial list appears, you can change your button selection to show the other list based on the same search criteria.

Name	Type	Modifier	Modified
TMR#1001	Problem Report	harvest	11/16/2000 12:44:19
TMR#1002	Problem Report	harvest	11/16/2000 1:17:54

Form Type—The Form Type drop-down list allows you to specify what kind of form to display in order to filter by its fields. By default, ten choices are available: Any Type, Application Change Request, Comment, Defect Tracking, ESD Change Request, Modification Request, Problem Report, Q and A, Testing Info, and User Contact. If you have added custom form types to Harvest, the custom form types also appear on this list. When the specified form is displayed, you can query on the various fields of that form.

If you select a form type from the Form Type drop-down list, the fields on that form type become available for use as filters. These filters vary depending on the form type chosen.

You can enter a value in any field to include only forms with that value in the current filtering operation. This feature allows you to locate specific information.

You do not need to know the precise value for a field to use it as a filter. You can use asterisks (*) in any field for wild card matching. For example, you might want to search for all Problem Reports that mention performance anywhere in the problem description. To do this, simply enter ***performance*** in the Problem Description field. Harvest returns all matching Problem Reports to the Form dialog's list.

In general, only trailing wild cards can be used in short text fields such as Category and Hardware on the Problem Report form. For longer fields such as Problem Description, you can use any number of wild cards (*) in any position for multiple character matching. The question mark (?) is also supported for single character matching.

Note: Form fields in the Find Form dialog are not case-sensitive. Information is stored exactly as it is entered, but can be searched on regardless of case.

Find—When the Find Form dialog is initially displayed, its list is empty. You must click the Find button for the current filtering options to take effect. Once Find is clicked, the forms matching the filtering criteria are displayed already selected in the list.

Note: If you have selected a filter but not clicked the Find button, the Form Report still shows the previous parameters as being in effect. It simply reports what is currently displayed on the Find Form dialog.

Find Package

The Find Package dialog allows you to execute complex filtering operations to locate packages with common attributes. After locating packages, you can execute available processes by right-clicking a package in the list box and choosing the process from the shortcut menu. For example, after locating all packages awaiting your approval, you could execute the approve process by right-clicking each package and choosing the approve process from the shortcut menu to invoke the Approve Process Execution dialog.

The shortcut menu also includes View Form, Save List to File, Delete and Properties options.

- Choosing View Form lists the associated forms. You can choose a form from the list to open the form in the form viewer, allowing you to view or modify the form.
- Choosing Save List to File opens a dialog that enables you to save your current search results to a specified name and location.
- Choosing Properties opens the Package Properties dialog for the selected package.

Note: If you click the Find button without specifying search criteria, all packages in Harvest are displayed in the dialog list.

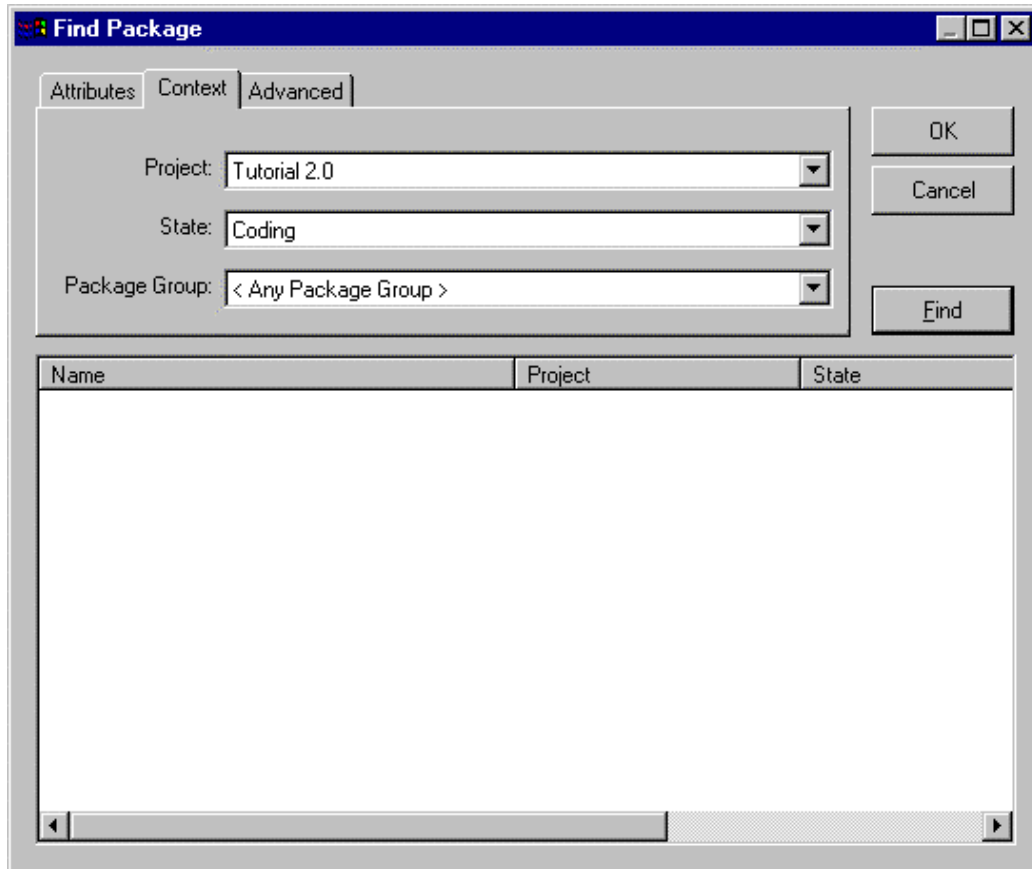
The screenshot shows the 'Find Package' dialog box. It has a title bar with the text 'Find Package' and standard window controls. Below the title bar are three tabs: 'Attributes', 'Context', and 'Advanced'. The 'Attributes' tab is selected. Under this tab, there are three search criteria: 'Name:' with a text box containing an asterisk (*), 'Assignee:' with a dropdown menu showing '< Any Assignee >', and 'Created By:' with a dropdown menu showing '< Any User >'. Below these is a 'Note:' text box containing an asterisk (*). To the right of the search criteria are three buttons: 'OK', 'Cancel', and 'Find'. At the bottom of the dialog is a list box with three columns: 'Name', 'Project', and 'State'. The list box is currently empty and has a scrollbar at the bottom.

Name—The Name field allows you to filter packages according to a naming pattern. You can use any number of wild cards (*) in any position for multiple character matching. The use of the question mark (?) is also supported for single character matching.

Assignee—This field allows you to locate packages assigned to a user. Use the drop-down list to select a user name, from all Harvest users, for filtering.

Created By—You can locate packages that were created by a specific user by selecting a user name from the drop-down list.

Note—This field allows you to locate packages according to text string values you enter in the format <string1> <string2>. The quotation mark (") can be used to indicate a space.

The image shows a Windows-style dialog box titled "Find Package". It has three tabs: "Attributes", "Context", and "Advanced", with "Advanced" currently selected. Inside the "Advanced" tab, there are three drop-down menus: "Project" (set to "Tutorial 2.0"), "State" (set to "Coding"), and "Package Group" (set to "< Any Package Group >"). To the right of these menus are three buttons: "OK", "Cancel", and "Find". Below the input fields is a table with three columns: "Name", "Project", and "State". The table is currently empty. At the bottom of the dialog is a horizontal scrollbar.

Project—To search for packages located in only one project, use the drop-down list to select a project for filtering. You can select the <Any Project> option to locate packages in any project.

State—If a project has been selected, you can filter for packages located within a state of that project. Use the drop-down list to select a state for filtering. You can select the <Any State> option to locate packages in any state.

Package Group—If a project has been selected, you can display only packages that belong to a package group within that project. Use the drop-down list to select a package group for filtering. You can select the <Any Package Group> option to locate packages in any package group. You can also use this filter in conjunction with a state to show only packages within the state of a project that belong to a package group.

Name	Project	State
------	---------	-------

The Advanced tab allows you to specify a date range and an approve status for the search.

Packages Created Before—Checking this box enables the date feature. Use the calendar to specify a date to locate packages created before that day.

Packages Created After—Checking this box enables the date feature. Use the calendar to specify a date to locate packages created after that day.

Waiting Approval by—You can locate packages that are pending approval by a specific user by selecting a user name from the drop-down list.

Approved Status—These buttons enable you to locate packages based on their status. Select **one** of the buttons, the default is Any.

- **Approved** locates packages based on the user name for whom packages have been approved.
- **Not Approved** locates packages based on the user name for which packages are awaiting approval.
- **Any** locates packages based on the user name for which packages have been approved and are awaiting approval.

Find or Select Version

Note: The title bar on this dialog varies according to how the dialog was invoked. If the dialog is invoked from the Tools menu, the title is Find Version. If the dialog is invoked from another dialog, the title is Select Version. Both dialogs function identically.

The Find Version dialog allows you to select item versions from a list. The various options on this dialog work in conjunction with one another to allow you to select versions according to multiple criteria. In addition, you can use the Find Version dialog simply to obtain version information.

When the Find Version dialog is initially displayed, its list box is empty. You must click the Find button for the current filtering options to take effect. Once Find is clicked, the versions matching the filtering criteria are displayed and selected in the list box. Click the OK button to return them to the calling dialog.

Note: If you have selected a filter but have not clicked the Find button, the Versions list still shows the previous parameters as being in effect.

Versions of items for which you do not have view access are not visible in the version list generated in the Find Version dialog. Repositories for which you do not have view access **are** visible in the version list, but cannot be opened, and their contents are not shown during recursive searches.

You can execute processes from the Find Version dialog. After you locate versions, right-click a version in the list box to open a shortcut menu with available processes for the context. You can choose a process to open its process execution dialog.

The shortcut menu also includes Print and Save options. Choosing Save opens a dialog that enables you to save your current search results to a specified name and location.

The appearance of the Find Version dialog is altered if the current state is associated with a snapshot view. The kind of information maintained for snapshots is different for working views, so when the Find Version dialog accesses versions in a snapshot view, some filtering options do not apply and are unavailable.

- The Tag option is set to No Tag and cannot be changed.
- The Branch option is set to Trunk Only because snapshots do not include any branch information, and it also cannot be changed.
- The Package option is disabled.
- The only possible choices for Version are Latest in View and All in View.

Find Version

Option: Name & View | Option: Date | Context setting

Item Name: *

Package: TMR#1001 ...

User: < Any User >

Item: All

Version/View: Latest in View

Version Tag: All

Branch: Trunk Only

OK

Close

Find

View Path: \

☐ Recursive

Name	Path	Version	Tag	Package
------	------	---------	-----	---------

Item Name—This field allows you to filter versions according to a naming pattern. You can use any number of wild cards (*) in any position for multiple character matching. The use of the question mark (?) is also supported for single character matching.

User—The User field allows you to filter versions based on the name of the user who created them. The drop-down list lists all Harvest users. You can select a user name for filtering from this list.

The Version filter affects the way that the User filter operates. When the User filter is used with the Latest in View Version filter, the Find Version dialog lists the latest version in that view only if the specified user created it; if the user did not create this version, no version is listed. When the Version filter is set to Latest, the latest version created by the user is displayed. For All and All in View, only the versions created by the specified user that match the other filtering criteria are listed.

Package—The Package field allows you to filter versions based on the name of the package that created them. Clicking the button next to the Package field displays the Select a Package dialog that lists all packages in the current context. This filter is not available when viewing versions in snapshot views.

The Version filter affects how the Package filter operates. When Latest in View is chosen, the items selected are all those that have changes associated with the package, but the version displayed for each item is the actual latest in the view, even if this version was not created by the package. This is to prevent overwriting changes when performing package check out.

When you use additional settings for the Version filter, the Package filter operates similarly to the User filter. When the Version filter is set to Latest, the latest version created by the package is displayed. For the All and All in View Version filters, only the versions created by the specified package that match the other filtering criteria are listed.

Item—The Item drop-down list allows you to filter versions on the basis of the modification status of items:

- **Modified** is the default and displays only versions for items that have been modified in this project.

For some processes, such as delete version and merges, initial versions cannot be used. Selecting Modified for this filter excludes all items for which only initial versions exist. This filter is also useful if you are checking out only changed items and want to prevent unchanged ones from being displayed.
- **Not Modified** displays versions for items only if the item has not been modified in this project. This is the initial version, or version 0, of items for which no other versions exist.
- **All** displays versions for all items regardless of whether they have been modified.

Version/View—This drop-down list allows you to filter the displayed versions in one of four ways:

- **All in View** displays all versions matching the other filtering criteria that exist in the view associated with the current state. Any later versions of an item in another view that have not yet been promoted to the current state are not displayed. This option is incompatible with viewing branch versions because they are not in a view. When selected, the only Branch option available is Trunk Only.
- **Latest in View.** This default option narrows the versions displayed to the latest for each item in the current view that match the other filtering criteria. Only one version per item is displayed. This option is incompatible with viewing branch versions because they are not in a view. When selected, the only Branch option available is Trunk Only.
- **All** displays all versions, including branch versions and trunk versions that do not yet exist in the view associated with the current state, that match the other filtering criteria. This option is not available when viewing versions in snapshot views.

- **Latest.** This option narrows down the versions displayed to the latest for each item that match the other filtering criteria, including those that do not yet exist in the view associated with the current state. Only one version per item is displayed. This option is not available when viewing versions in snapshot views.

Version Tag—This drop-down list allows you to filter the displayed versions in one of six ways:

- **All** displays versions that match the other filtering criteria regardless of whether they are tagged.
- **No Tag** displays only versions that do not have a tag associated with them and match the other filtering criteria.
- **Reserved** displays only versions tagged as reserved (R) that match the other filtering criteria.
- **Merged** displays only versions tagged as merged (M) that match the other filtering criteria.
- **Removed** displays only versions tagged as removed (D) that match the other filtering criteria.
- **Any Tag** displays only versions that have been tagged as reserved (R), merged (M), or removed (D), and match the other filtering criteria.

Branch—This drop-down list allows you to filter the displayed versions in one of the following ways:

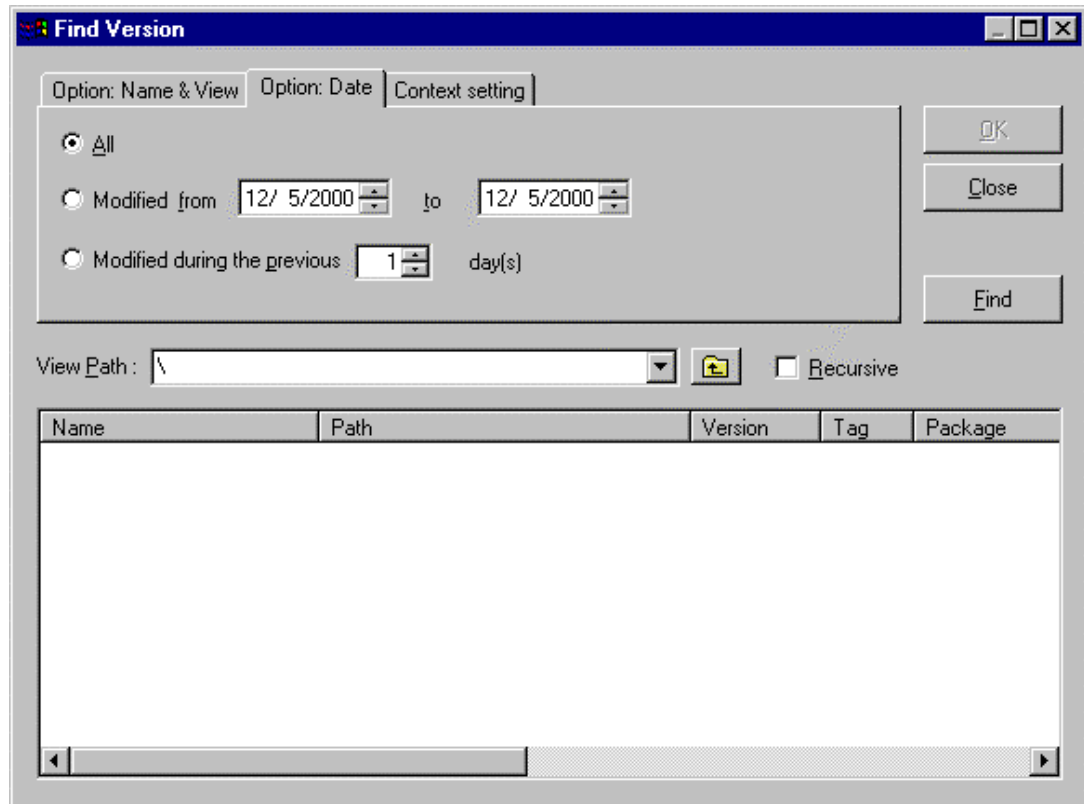
- **Trunk Only** is the default and displays versions on the trunk in this project that match the other filtering criteria.
- **Trunk and Branch** displays all versions that match the other filtering criteria.
- **Branch Only** displays all branch versions that match the other filtering criteria. This option is incompatible with the All in View or Latest in View Version filters because branches do not exist in a view.
- **Unmerged Branch** displays all unmerged branch versions that match the other filtering criteria. This option is incompatible with the All in View or Latest in View Version filters because branches do not exist in a view. Use this option to select versions before executing the concurrent merge process.

View Path—This drop-down list lets you quickly change the current location to any position within the current path. When this menu is popped up, the current path is split into individual menu items. Selecting any menu item causes your position to change to the selected path immediately. Clicking the button at the right of the drop-down list moves up one directory in the current path.

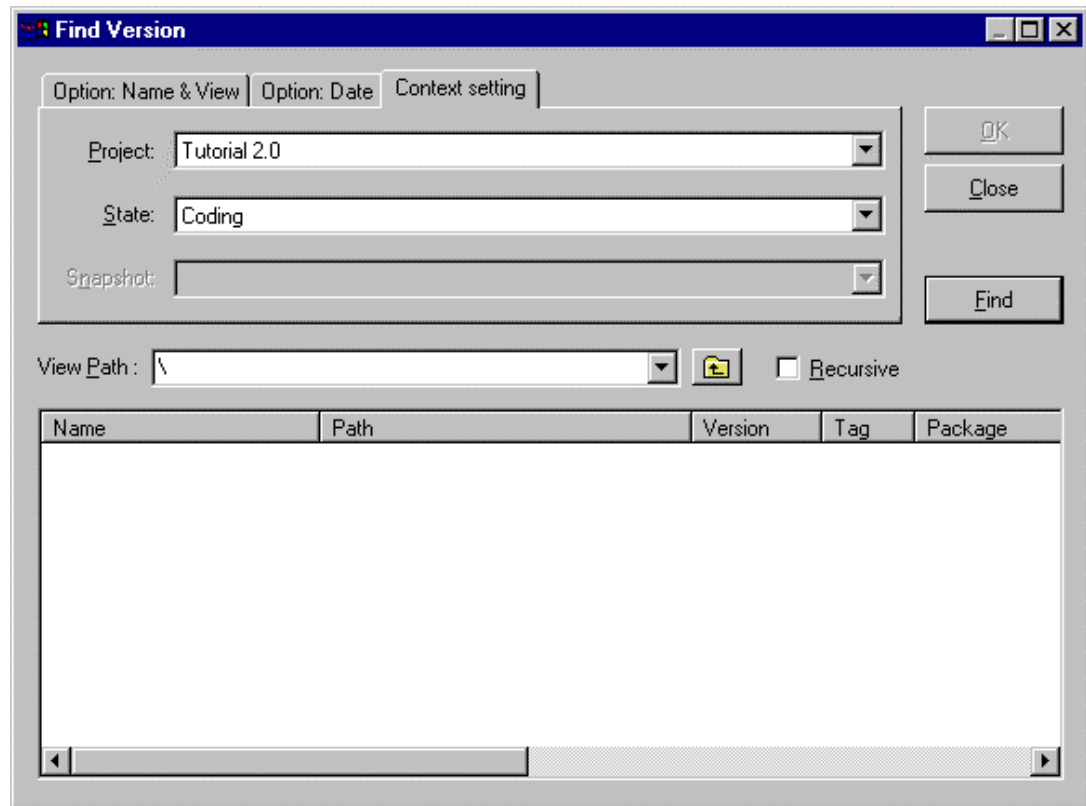
Recursive—You can use the Recursive option to display versions that match the other filtering criteria in every path below the current one. You can then select versions from multiple paths to be included in this operation.

When you select Recursive and click the Find button, Harvest expands each path so that the item names displayed in the list box include relative paths from your current position.

Note: Selecting the Recursive option disables the View Path drop-down list. To change your position after filtering recursively, deselect Recursive before you try to move.



The Option: Date tab allows you to specify a date range for the search. By default, the all files button is selected, indicating date checking will not be performed. Checking the modified between or modified during the previous days button enables you to select different date searches that can be performed. You can select to search between two dates or search the previous x number of days.



Project—To search for versions located in only one project, use the drop-down list to select a project for filtering.

State—If a project has been selected, you can filter for versions located within a state of that project. Use the drop-down list to select a state for filtering.

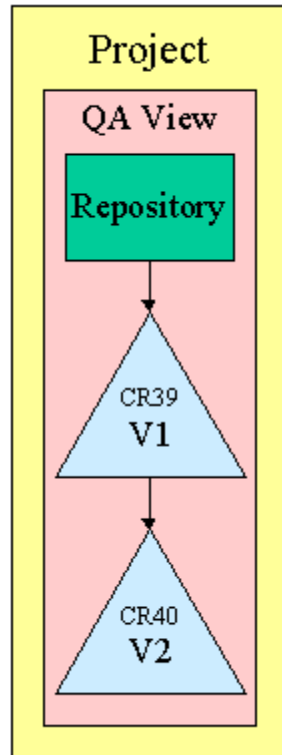
Snapshot—If a snapshot has been selected, you can filter for versions belonging to the snapshot. Use the drop-down list to select a snapshot for filtering.

Using Multiple Filtering Criteria

To use the Find Version dialog effectively, it is helpful to understand how the various filtering criteria interact. You can use the Item Name, User, Package, and Recursive search parameters to generate a list of items, with version information specified based on the Version, Branch, and Tag parameters.

The Latest in View Version option has a special meaning in relation to the Package or User filters. If Latest in View is selected, and a Package is also selected, the search returns the latest version of all items that have at least one version associated with that package.

Consider the following illustration. In this figure, two packages named CR39 and CR40 have just been promoted to the QA state in the QA view. The delta tree for one item is shown in the figure below. This item was changed by both packages.



In QA, a user checks out the item using the Latest in View option with CR39 as the associated package. The Package filter selects this item because it has been changed by CR39, but the Version filter selects version 2 of the item. This is because the version filter selects the latest in view regardless of the package. As the latest version in the QA view, version 2 of the item, which includes the changes made for CR40 and CR39, is checked out.

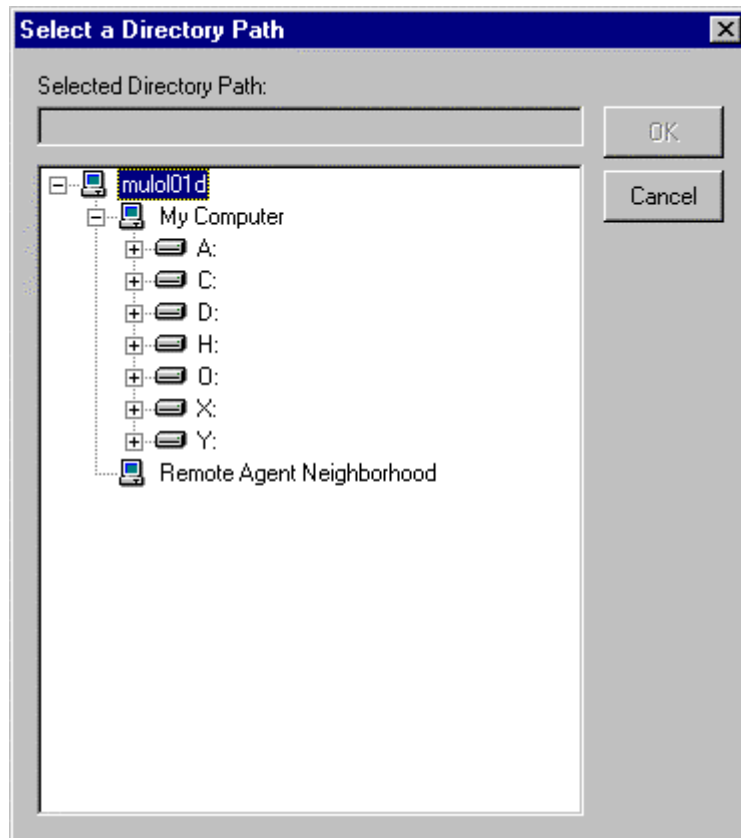
The following table shows the results of using the Find Version dialog using various filtering criteria typically used for the check in and checks out processes. This is a partial listing of the many different possible combinations.

Item	Version/View	Package	Results
Modified and Not Modified	Latest in View		Latest version of all items in view.
Modified and Not Modified	Latest in View	Package	Latest version of all items in view where at least one version is associated with the selected package.
Modified and Not Modified	All in View		All versions of all items in view.
Modified and Not Modified	All in View	Package	All versions of all items in view where at least one version of each item is associated with the selected package.
Modified and Not Modified	All		All versions of all items in the selected project.
Modified and Not Modified	All	Package	All versions of all items in the selected project where at least one version of each item is associated with the selected package.
Modified and Not Modified	Latest		Latest versions of all items in the selected project.
Modified and Not Modified	Latest	Package	Latest versions of all items in the selected project where at least one version of each item is associated with the selected package.
Modified	Latest in View		Latest versions of modified items in view.
Modified	Latest in View	Package	Latest versions of modified items in view where at least one version of each item is associated with the selected package.
Modified	All in View		All versions of modified items in view.

Item	Version/View	Package	Results
Modified	All in View	Package	All versions of modified items in view where at least one version of each item is associated with the selected package.
Modified	All		All versions of modified items in the selected project.
Modified	All	Package	All versions of modified items in the selected project where at least one version of each item is associated with the selected package.
Modified	Latest		Latest versions of modified items in the selected project.
Modified	Latest	Package	Latest versions of modified items in the selected project where at least one version of each item is associated with the selected package.

Select a Directory Path

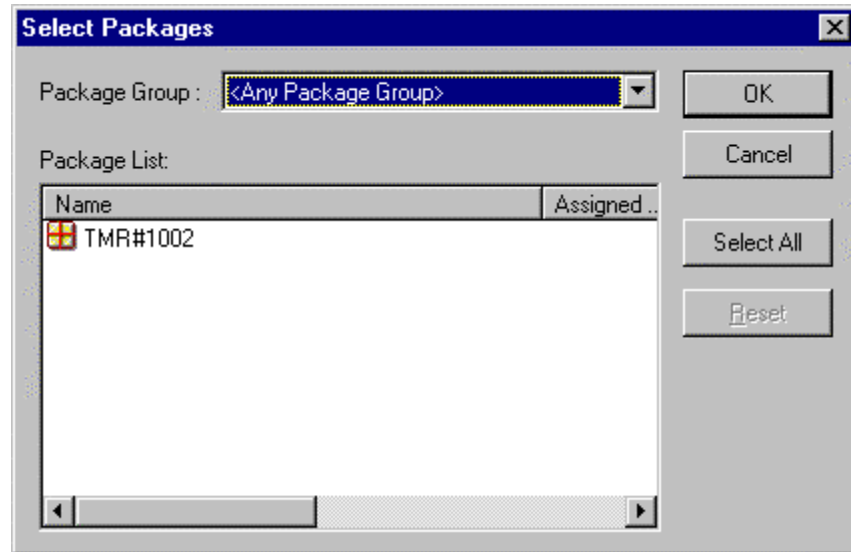
The Select a Directory Path dialog lists directories in the current client directory. You can traverse through the file system by double-clicking a directory or you can choose a directory by selecting it and clicking the OK button.



Select Packages

The Select Packages dialog lists the package groups and packages in the current project and allows you to select a package to populate the Package field in the calling dialog.

The Select Packages dialog is invoked from the package-related process execution dialogs such as approve and check out, and the Find Version dialog.

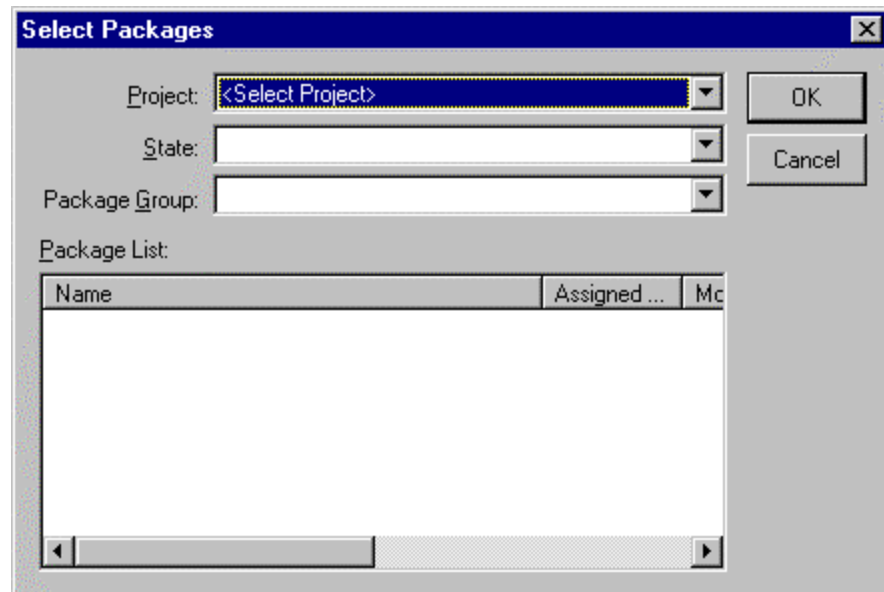


The Package Group Filter field allows you to filter packages according to a package group. Use the drop-down list to select a package group in your current project. If you select a package group, its packages are listed in the Packages list. You can select a package and then click the OK button.

Select Packages (Cross Project)

The Select Packages (cross project) dialog allows you to select the project, state, or package group from which packages are to be merged (pulled) into your current project. The packages you select populate the Package field in the calling dialog.

The Select Packages dialog is invoked from the Cross Project Process Execution dialog.



Project—You can select a project from which you can select the packages to be merged into your current project by using the drop-down list to list available projects.

State—If a project has been selected, you can select a state from which you can select packages to be merged into your current project by using the drop-down list to list available states.

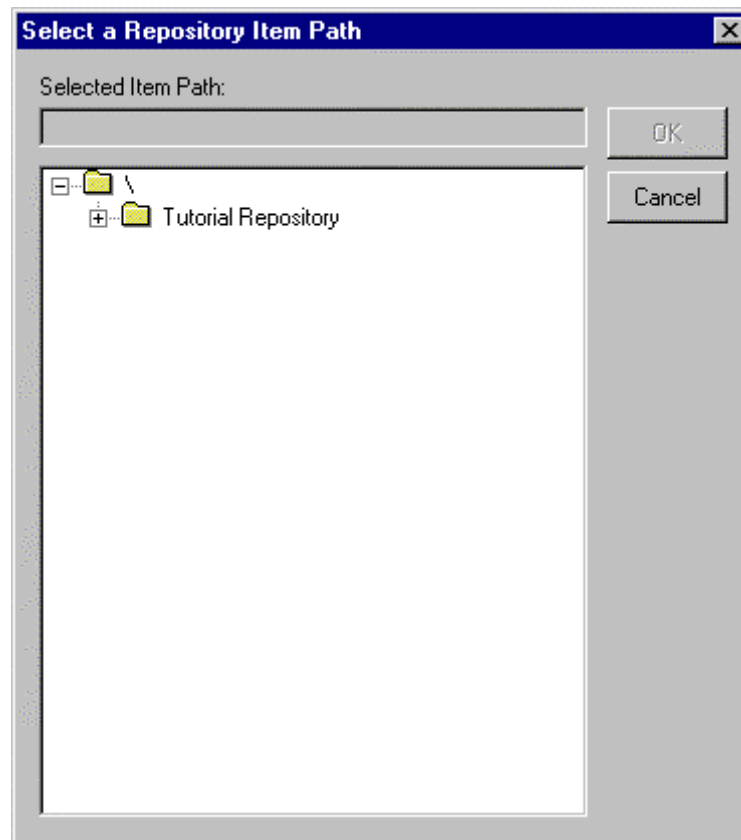
Package Group—If a project or state has been selected, you can select a package group from which you can select packages to be merged into your current project by using the drop-down list to list available states.

After choosing your filtering criteria, available packages are listed in the Package List field. The packages you select in this field are returned to the Cross Project Process Execution dialog after clicking the OK button.

Select a Repository Item Path

The Select a Repository Item Path dialog lists paths in the current path. You can traverse the paths in a repository by double-clicking a path name; or you can choose a path by selecting it and clicking the OK button.

The Select a Repository Item Path dialog is invoked from the Check In and Check Out Process Execution dialogs and the Load Repository dialog.



Select User

When adding users to a new user group, the Select User for User Group dialog allows you to locate and select users from a list.

When the Select User dialog is initially displayed, the From User Group(s) field lists all user groups and the Select User(s) list is empty. After searching for users, the Select User(s) list shows the results and you can return the users to the calling dialog.

You can search for users to populate the Select User(s) list in these ways:

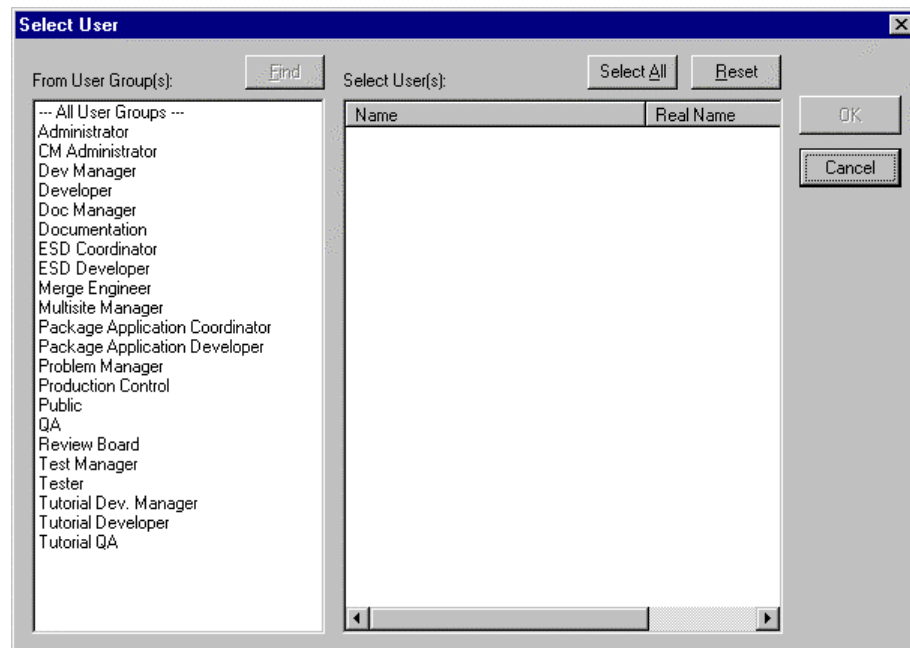
- To locate the users that belong to a specific user group, select the user group and click the Find button. This lists the users that belong to that group in the Select User(s) field.
- To locate all users in all user groups, select All User Groups in the From User Group(s) list and click the Find button. This lists the all users in all user groups in the Select User(s) field.

Note: If you have selected a user group but have not clicked the Find button, the Selected User(s) list still shows the previous selection.

Select one or more users and click OK to return the users to the calling dialog.

To return all users in the Select User(s) list to the calling dialog, click the Select All button and then OK.

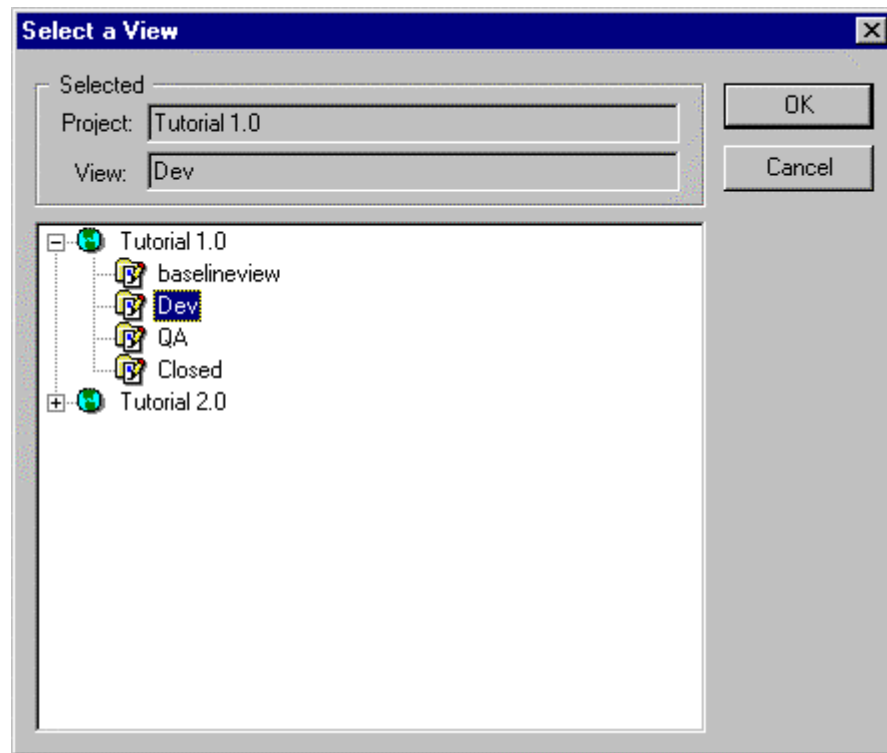
The Select User dialog is invoked from the User Groups Properties dialog.



Select a View

The Select a View dialog lists the available views in Harvest, and allows you to filter views according to project, and once a project has been chosen, you can select a view.

The Select a View dialog is invoked only from the Compare Views Process Execution dialog. After the view is selected, it is returned to the view field in the calling dialog.



Selected—Your selected project and view are displayed.

View List—This list box displays the views in the currently selected project. To locate a view, navigate the hierarchy tree. To select a view, double-click it or select it and click OK.

ISPF Client Interface

ISPF is a standard NONGUI menu driven interface to the OS/390 operating system and file utilities. The Harvest ISPF interface acts as an extension to this environment. It offers a subset of functions that enable OS/390 users to check data into and out of the Harvest database without requiring them to leave their ISPF environment.

Log In

When you invoke the ISPF Harvest interface, the following messages are written to the screen prior to the display of the Login panel appears.

```
Connecting to project <platinum> on <130.200.28.16> RTserver.  
Using local protocol.  
Could not connect to <130.200.28.16> RTserver.  
Connecting to project <platinum> on <130.200.28.16> RTserver.  
Using tcp protocol.  
Message from RTserver: Connection established.  
Start subscribing to subject </pt_HClient://USILCA11/16777339>.  
***
```

You can set an option to retain the Harvest user name between sessions or set a default value to be used each time you enter Harvest. This can be the TSO userid. If the userid you enter on the login panel matches the TSO userid, you do not need to enter the password.

The broker name is the machine on which the Harvest server is running. This value can be retained between sessions. You can set the option to retain this value or set a default value to be used for every log in. See the section "Settings" for more information.

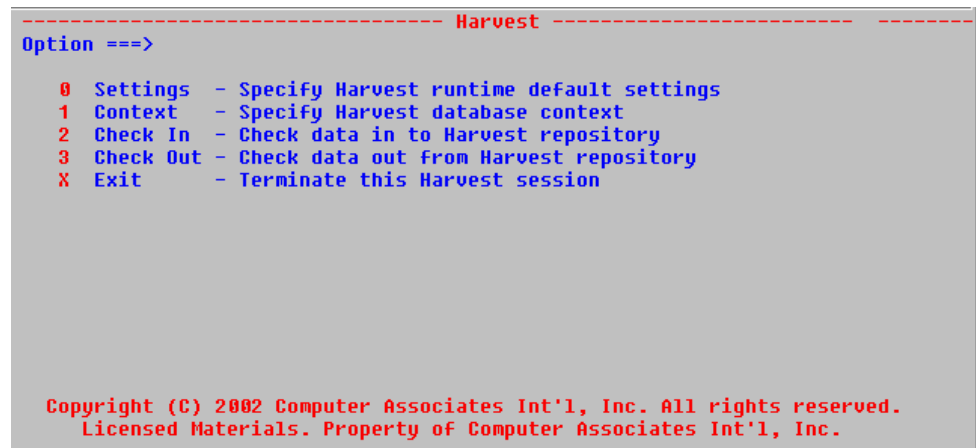
Enter the TCP/IP address to connect to the broker.



After a successful login, a valid context must be set before the Harvest primary panel is displayed. The interface displays one or more context settings panels. You must select valid context settings on each panel in order to continue. See the section "Harvest Context" for setting your context.

Harvest ISPF Primary Panel

The primary panel offers selections for settings, context, check in, and check out. The primary panel is shown below.



Settings

The settings panel enables you to set usage values once, and they remain in effect until they are changed.

```

Harvest Runtime Default Settings
Command ==>

Usage (Default/Last/None)    Settings
D/L/N . . L Harvest User . . 5
D/L/N . . L Broker Name . . 1
D/L/N . . L Project . . . 1
State . . . . . 1
View . . . . . 1
Package . . . . . 1
Item Path . . . . . 1
CI Process . . . . . 1
CO Process . . . . . 1
CI Options . . . Mode UR/RO/UK . . UR
                        Path P/PC/A . . PC
                        Filter NE/N/E . . NE
                        Delete Y/N . . N
                        Comments Y/N . . N
CO Options . . . Mode U/C/B/R . . U
                        Replace Y/N . . Y
Confirmation CI/CO/B/N . . B (Check In/Check Out/Both/None)
Dynamic Status Log Y/N . . Y
Interactive or Batch I/B . . I

```

Harvest Context

The context consists of the Harvest project, state, view, package, and item path. A valid context must be set prior to executing a check in or check out. You set these values once, and they remain in effect until they are changed. Selecting option 1 displays a window that allows you to set the context and map ISPF processes to a Harvest process defined for that context.

Setting Context

You can display the context panel by selecting 1 on the primary panel. The context window allows you to set the context and map ISPF processes to a Harvest process defined for that context. You can modify any of the context settings by entering the new information in the value field. You can also request a selection list by entering a wildcard specification, or by blanking out the current information. From the list, select the object you want by entering **S** beside the object name.

If you change one context setting, the remaining settings are validated. A list automatically appears to select a valid context object when an object being validated cannot be found in the repository.

```

Option ==> 1

Current Context Settings

Command ==>
Project . . . Simple Test Project
State . . . Development
View . . . Development
Package . . . MR- 001
Item Path . .
CI Process . . Check In Items
CO Process . . Check Out for Update

Copyright (C) 2002 Computer Associates Int'l, Inc. All rights reserved.
Licensed Materials. Property of Computer Associates Int'l, Inc.

```

Setting the Project

Project selection is optional. The following panel appears when a list of projects is requested. You can make your selection from this list or enter **Cancel** at the Option prompt to cancel the project list processing.

```

Harvest
Option ==> 1

┌────────── Current Context Settings ──────────┐
│ Command ==>                                  │
│ Project . . .                               │
│ State . . . Development                     │
│ U ─────────── Select One Project ───────────┐
│ P ┌──────────┐                               │
│ I │ Command ==>                                Row 1 of 1
│ C │                                     Scroll ==> HALF
│ C │ Project Name                             Message
│   │                                         │
│   └────────── Simple Test Project ─────────┘
│ ***** Bottom of data *****
Copyr
Li

```


Setting the Package

Once you set the state, you can select the package. The following panel displays a list of packages in the current state and you can make your selection from this list.

```
----- Harvest -----  
Option ==> 1  
  
┌────────── Current Context Settings ──────────┐  
│ Command ==> │  
│ Project . . . Simple Test Project │  
│ State . . . Development │  
│ View . . . Development │  
│ Package . . . │  
│ Item Path . . . │  
└────────── Select One Package ───────────┘  
C  
C  
│ Command ==> Row 1 of 2  
Scroll ==> HALF  
  
Copyr | Package Name Message  
Li | MR- 001  
MR- 002  
***** Bottom of data *****
```


Check In Process Selection

Once you set the state, you can select the Harvest check in process to be used by the ISPF interface. A list shows all Harvest check in processes from the current state. Select the check in process to be used by ISPF by entering an **S** next to the selected process.

```
----- Harvest -----
Option ==> 1

      Current Context Settings
      |
      | Command ==>
      | |----- Select Harvest process to be mapped to CheckInProcess -----|
      | P |
      | S | Command ==>
      | U |
      | P | Harvest Process
      | I |
      | C | s Check In Items
      | C | ***** Bottom of data *****
      |
      |
Copyr |
Li |
```


The Check In Panel

```
Harvest Check In
Command ==>

From ISPF Library:
Project . . .
Group . . .
Type . . .
Member . . . (Pattern for list, blank or "*" for all members)

From Other Partitioned or Sequential Data Set:
Data Set Name . . . 'ASBDEV.TEST.TESTDATA.Fb.cpp'

To Harvest Context:
Project . . . Simple Test Project
State . . . Development
Package . . . MR- 001
View . . . Development
Item Path . . simple repository
Process . . . Check In Items

Options:
Mode UR/RO/UK . . UR Path P/PC/A . . A Filter NE/N/E . . NE
Delete Y/N . . N Comments Y/N . . N
Interactive or Batch I/B . . I
```

From

You must enter the PDS member of sequential file to be checked in.

Context

These fields display the current context and can be modified.

Options

The check in function operates in one of three modes, governed by the state of the Mode:

- **UR (Update and Release).** The new item or version is checked in to the repository and, if the item was reserved by a package during a previous check out, the reserved tag is removed from the item.
- **RO (Release Only).** No check in is performed and the item is not updated, but the item is no longer marked as reserved for the current package.
- **UK (Update and Keep).** The item in the view is updated (or created) and the current package keeps it reserved.

The Path provides a variety of ways to check in files and filter the files being checked in. Select **one** of the following:

- **P (Preserve Directory Structure)** checks in the selected files to repository view paths with names that correspond to their client directory location, if these view paths currently exist.

- **PC (Preserve and Create Path Structure)** checks in selected files to paths with names that correspond to their client directory location, and creates any view paths that do not currently exist.
- **A (All Files to Same View Path)** checks in all selected files to the same path in the destination view, ignoring the client directory structure.

The Filter menu provides a variety of ways to filter the files being checked in. Choose **one** of the following:

- **NE (New or Existing Items)** checks in all selected files, if they are reserved by the package or did not previously exist.
- **N (New Items Only)** limits the check in to files that do not have corresponding items in the Harvest repository.
- **E (Existing Items Only)** limits the check in to files that have corresponding items reserved by the package. Any files without corresponding items are skipped. You can use this filter to prevent the existence of unwanted files, such as temporary files or templates, in your repository.

Note: Only the item filters enabled in the Check In Process Properties dialog are available for selection in this field.

The Delete option allows you to specify the deletion of files or members after a successful check in:

- Type Y to delete files or members.
- Type N to not delete files or members.

The Comments option allows you to enter a comment about the check in:

- Type Y to display a window in which you can enter a description.
- Type N to not display a comment window.

Select interactive or batch and then execute the check in. Results of the check in are displayed in a progress window.

Check Out

The check out panel appears if you select the check out function from the primary panel. It allows you to copy versions of items under Harvest's control to external directories where the items can be updated or browsed. Alternatively, items can simply be reserved without actually copying the data to a client directory. For a complete description of the check out process, see the chapter "Processes" in this guide. The check out panel is described below.

The Check Out Panels

```
Harvest Check Out
Command ==>

To ISPF Library:
Project . .
Group . . .
Type . . .

To Other Partitioned or Sequential Data Set:
Data Set Name . . . 'ASBDEV.TEST.TESTDATA.FB'

From Harvest Context:
Project . . . . Simple Test Project
State . . . . Development
Package . . . . MR- 001
View . . . . Development
Item Path . . . simple repository
Process . . . . Check Out for Update

Check Out Options:
Mode U/B/C/R . . U (Update/Browse/Concurrent/Reserve)
Replace Y/N . . N

Interactive or Batch I/B . . I
```

To

The To ISPF Library field allows you to specify the library to which you want to check out the PDS. The To Other Partitioned or Sequential Data Set field allows you to specify the member to check out. To display a member list, leave the Data Set Name field blank, from the member list type an S next to the member you want to select.

Context

These fields display the current context and can be modified.

Options

Four modes are available for the check out, select one:

- **U (Update)** copies selected versions to the destination client directory and creates a reserved version on each item's trunk, allowing the corresponding files to be checked back in. A valid project is required.
- **B (Browse)** copies the items to the destination directory but does not allow you to check the files back in. The Project field can be blank.
- **C (Concurrent Update)** copies selected versions to the destination client directory and creates a reserved version on a package branch for each item. All package updates accumulate on this branch. A valid project is required.
- **R (Reserve Only)** does not move any data to external directories but creates a reserved version with a reserved tag (R) on each item's trunk so that corresponding files can be checked in. A valid project is required.

The Replace option does not affect the check to sequential or PDS members. It is used when USS is implemented.

Select interactive or batch. If you select batch you are prompted for job card JCL and your job is submitted in batch mode.

The Data View panel appears showing a list of item paths, versions and items in the current context. Characters beside each line in the listing identify an item path (IP), version (V) or item (I). You can expand an item path or item by entering an **X** beside it: item paths expand to other item paths and items, items expand to versions.

```

Harvest Data View                                     Row 1 of 6
Command ==>                                           Scroll ==> PAGE

Project : Simple Test Project
State   : Development
View    : Development
Item Path: simple repository

Type Name                                           Message

IP  DOCS
IP  EXE
IP  OBJ
IP  SRC
X IP  TESTDATA
IP  vbproject1
***** Bottom of data *****

```

The panel below shows the item paths and an item expanded.

```
Harvest Data View                                     Row 1 of 9
Command ==>                                           Scroll ==> PAGE

Project : Simple Test Project
State   : Development
View    : Development
Item Path: simple repository

Type Name                                           Message

IP      DOCS
IP      EXE
IP      OBJ
IP      SRC
IP      TESTDATA
S IP     DXYTEST
IP      HPGLTEST
I       TEXTPS.C
IP      vbproject1

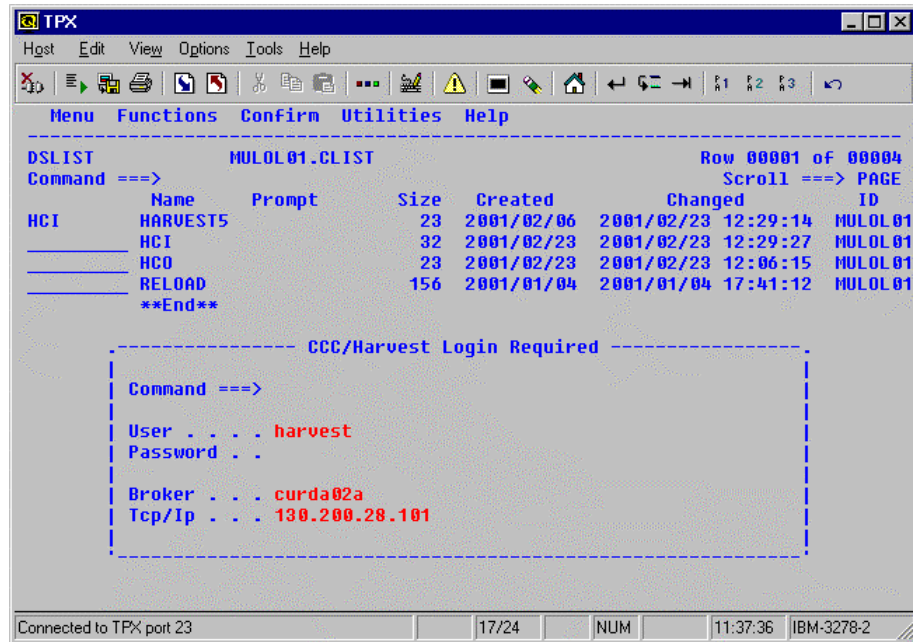
***** Bottom of data *****
```

Select an item or version to check out by entering an S beside it. Results of the check out are displayed in a progress window.

Integrating with ISPF Data Set List Utility (ISPF 3.4)

You can enter **hci** or **hco** to perform check ins or check outs to a PDS from the panel 3.4 member list. The login prompt appears followed by the checkin or checkout panels.

A member list on ISPF 3.4 with the Harvest login window looks like this:



ISPF handles each request by looking for an exec with the specified name. One parameter is passed to this exec, and it is the data set name with the member name in parenthesis.

You have to log in to Harvest each time the exec is called. The exec logs you in to Harvest, performs the requested operation, and then logs you out. This process is repeated each time.

After logging in the check out or check in panel is displayed.

Mapping

When defining the Harvest repository, you can choose one of two methods for mapping OS/390 file name conventions to the database hierarchy.

- OS/390: Item extensions are not supported and cause an error on check out. This mapping is best suited for OS/390 development. The PDS is checked into a single directory. Items are checked out to a PDS with the name <item path>(item). Sequential files are checked out to <item>, where <item path> is the PDS name. This mapping is used when the OS/390 option is enabled on the Repositories Properties dialog.
- Directory: File extensions are supported. This mapping is best suited for cross-platform development. Last level qualifiers are mapped to extensions; each qualifier is mapped to a directory.

On check in the last level qualifier becomes the item extension. The PDS member name becomes the item name.

On check out the item extension is appended as the last level qualifier of the PDS name. The item name becomes the member name. Directories that do not conform to OS/390 qualifier syntax cause an error on check out. This is the default mapping for repositories.

Using the Windows Extension (HarWind)

The Harvest Windows Extension (HarWind) provides a method for performing check in and check out functions from within the Windows Explorer. This allows you to execute these most common configuration management tasks without having Harvest running on your client machine.

The Harvest Windows Extension is available for use on Intel Windows NT 4.0 and Windows 2000 platforms.

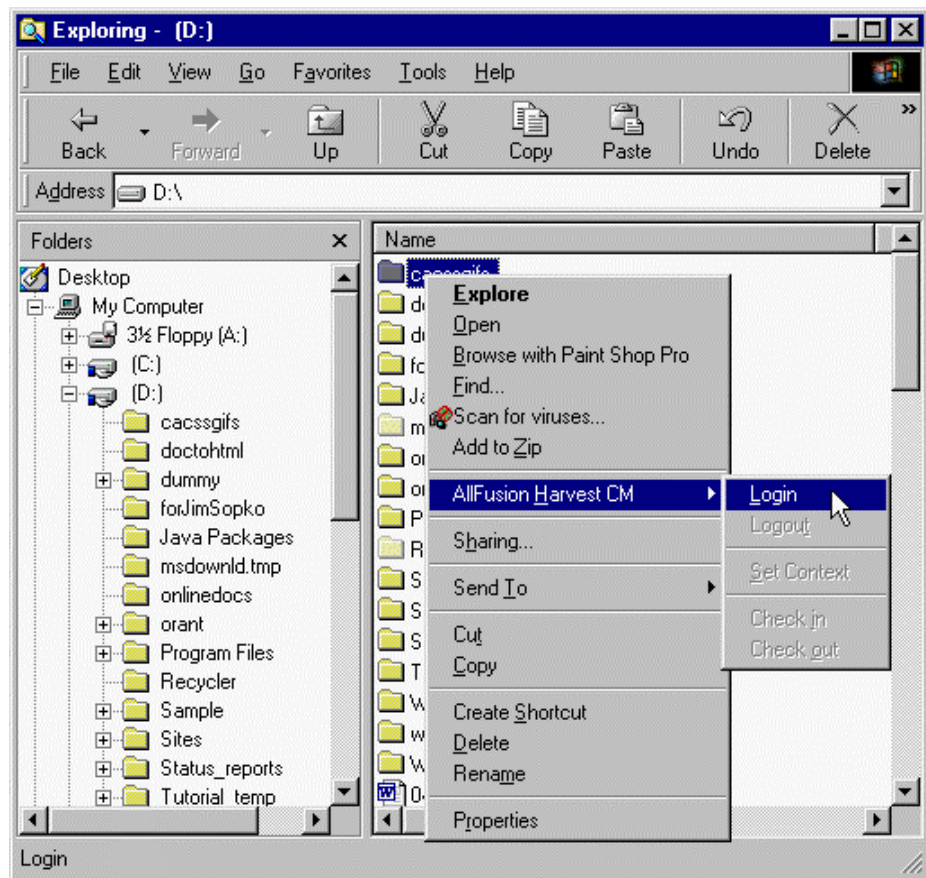
For detailed information on the check in and check out processes, see the chapter “Processes” in this guide.

Log In

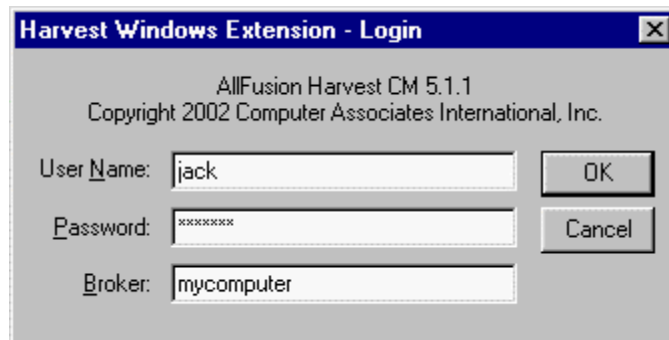
Before you can perform any check in or check out functions, you must first log in to Harvest. To do this:

1. Right-click any folder in your Windows Explorer.
A shortcut menu lists all of the actions that can be performed on the folder. Included in this menu is a Harvest option.
2. Choose AllFusion Harvest CM. Another shortcut menu lists all of the Harvest actions that can be performed.

Note: Initially some of the menu options are disabled until the Set Default Context option has been performed.



3. Select Login. The Harvest Windows Extension—Login window appears.
4. Enter your Harvest user name and password into the appropriate fields.



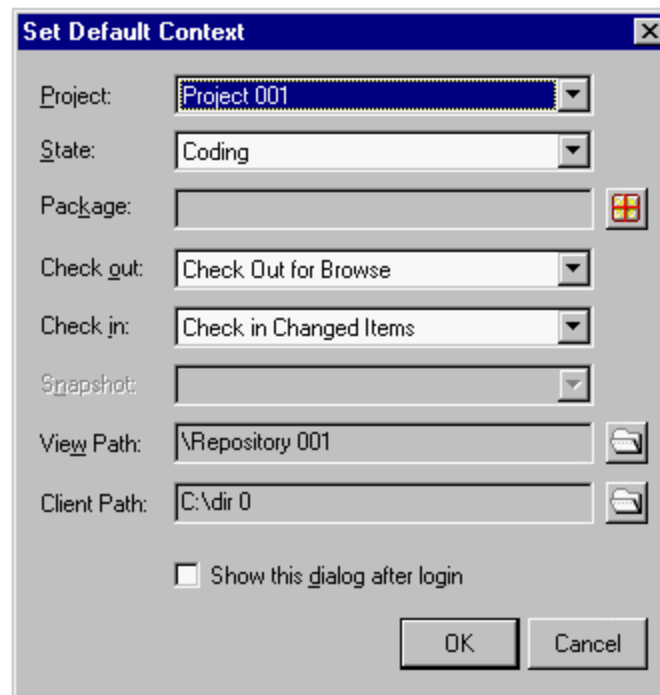
5. In the Broker field, enter the name of the machine where the Harvest broker is running.
6. Click OK or press <Enter>.

Note: Once you log in, you remain logged in until you execute the log out function. Closing the Windows Explorer does not log you out of the Harvest Windows Extension.

Set Default Context

After you have successfully logged in, and before you can perform any check in or check out functions, you need to set your working context. To do this:

1. Select the directory you want to use as your destination directory for checking out.
2. Right-click the directory, and then select Harvest, Set Context. The following window appears:

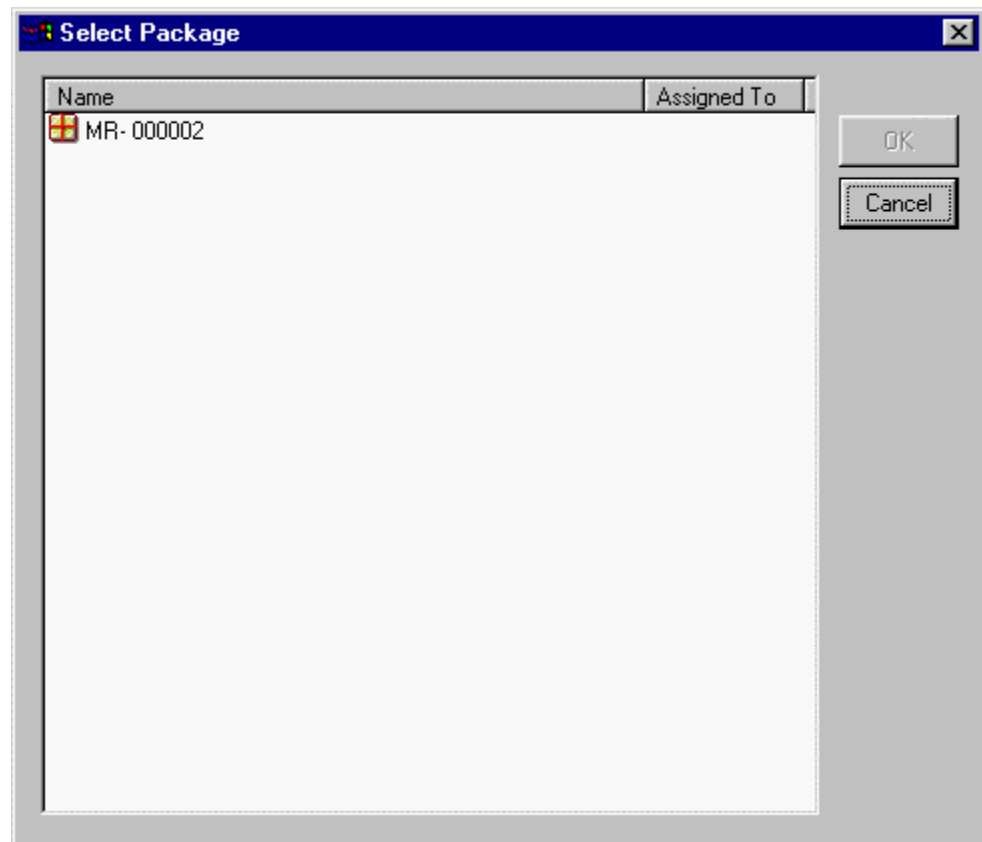
The image shows a Windows-style dialog box titled "Set Default Context". It contains several fields and controls: "Project:" with a dropdown menu showing "Project 001"; "State:" with a dropdown menu showing "Coding"; "Package:" with an empty text field and a small red button with a white grid icon; "Check out:" with a dropdown menu showing "Check Out for Browse"; "Check in:" with a dropdown menu showing "Check in Changed Items"; "Snapshot:" with a dropdown menu showing an empty field; "View Path:" with a text field containing "\Repository 001" and a folder icon button; "Client Path:" with a text field containing "C:\dir 0" and a folder icon button; a checkbox labeled "Show this dialog after login" which is currently unchecked; and "OK" and "Cancel" buttons at the bottom right.

Note: The settings in this window remain the same until you modify them. Once you set your context, you do not have to open the Set Context window again unless you want to change the settings.

Project—From the Project drop-down list, select the Harvest project you want to work in.

State—From the State drop-down list, select the state you want to work in within the project.

Package—Select the package that you want to use when performing check ins and check outs by clicking the button next to this field to open the Select Package dialog from which you can select a package in the current state.

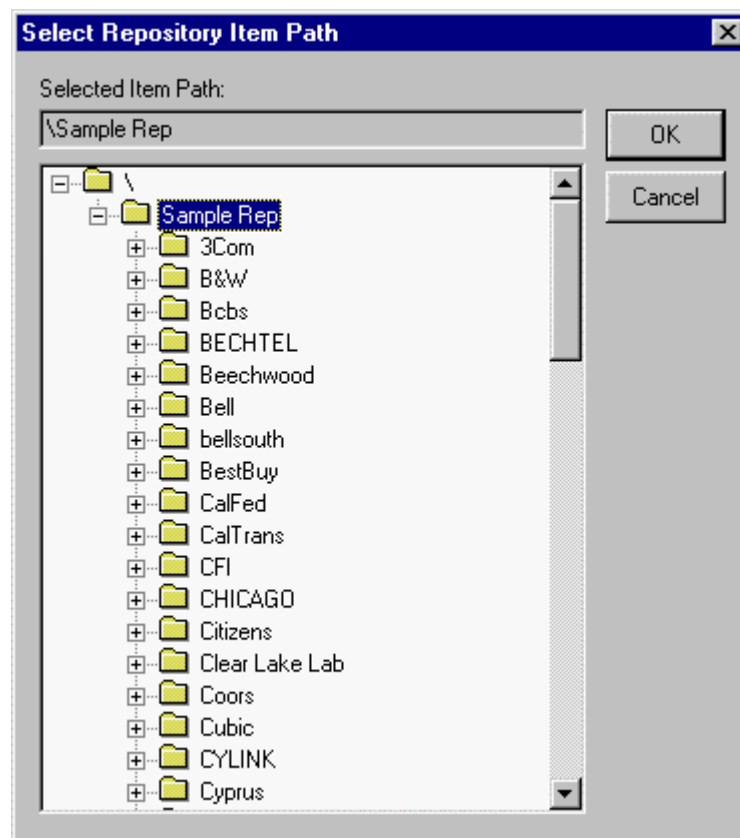


Check out—Each checkout process that is available in the selected state is listed in this option menu. Select the one you want to use as your default.

Check in—Each checkin process that is available in the selected state is listed in this option menu. Select the one you want to use as your default.

Snapshot—The Snapshot option menu is enabled only if the state you selected includes all snapshot views. Select which snapshot view you want to use as your default when checking out.

View Path—The view path specifies a location in the repository from which items are checked out or files are checked in. By specifying a view path, you are setting a default path that is placed in this field in subsequent check outs and check ins. To change your repository path, click the button next to the View Path field. This invokes the Select Repository Item Path dialog that allows you to browse through available paths and select a location.



Client Path—The directory selected for the Client Path field specifies the destination on the file system from which files are checked in or to which items are checked-out. If either of the preserve structure check out options is being used, the check out specifies a path and this specification appends to the file system directory. To change the destination, click the button next to the Client Path field. This invokes the Select a Directory Path dialog that allows you to navigate through directories available on your system to select a file.

Show this dialog after login – Selecting this checkbox causes the Set Context dialog to popup automatically after subsequent logins.

Click OK to save your settings, or Cancel to discard your settings.

Executing Check Out and Check In

Two ways are available to select files to be checked in or out. One is to simply select the files from the Explorer window; the other is to perform a Find for the files you need.

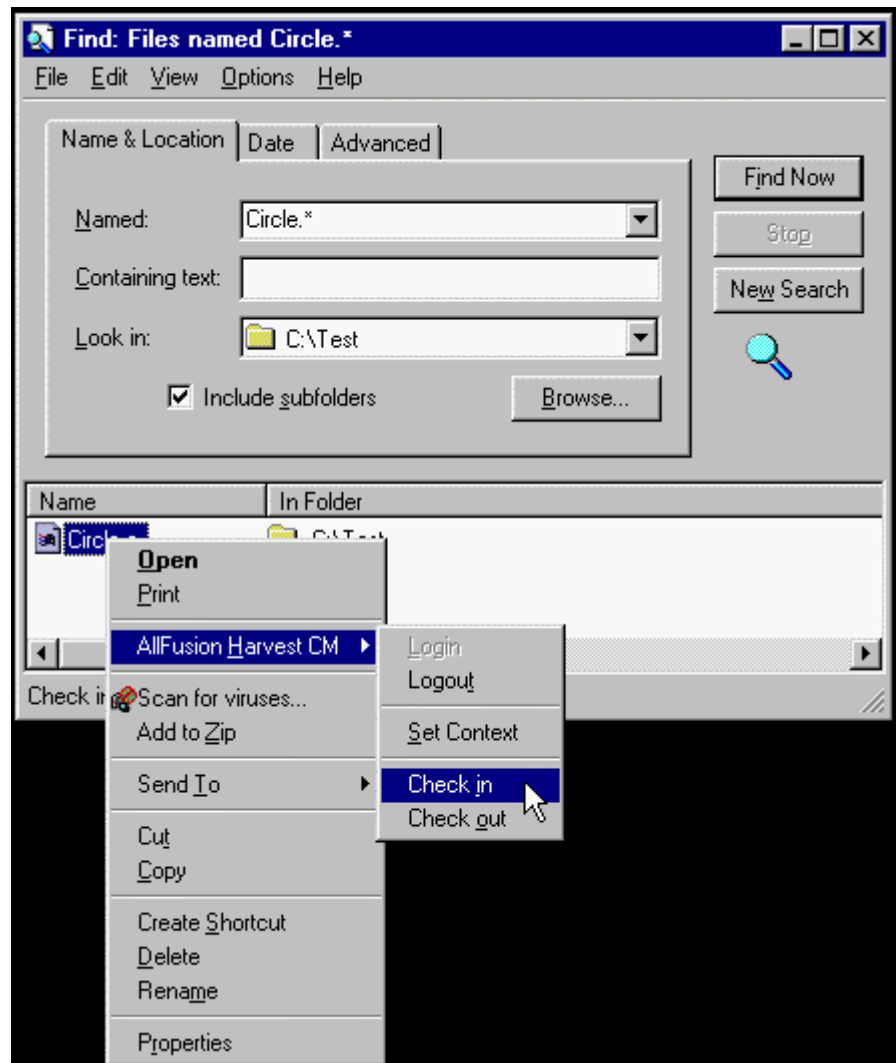
To check in or check out files from the Explorer window:

1. Select the files you want to check in or check out. Alternatively, you can select a directory, which checks in or checks out all of the files within that directory, and also includes all subdirectories.
2. Right-click one of the selected files, or the selected directory, and then choose AllFusion Harvest CM, *check in* or *check out* from the shortcut menu.
3. The check in or check out dialog displays and you can set options and execute the process.
4. You can see the status and results in the Message Log.

To check in or check out files from the Find window:

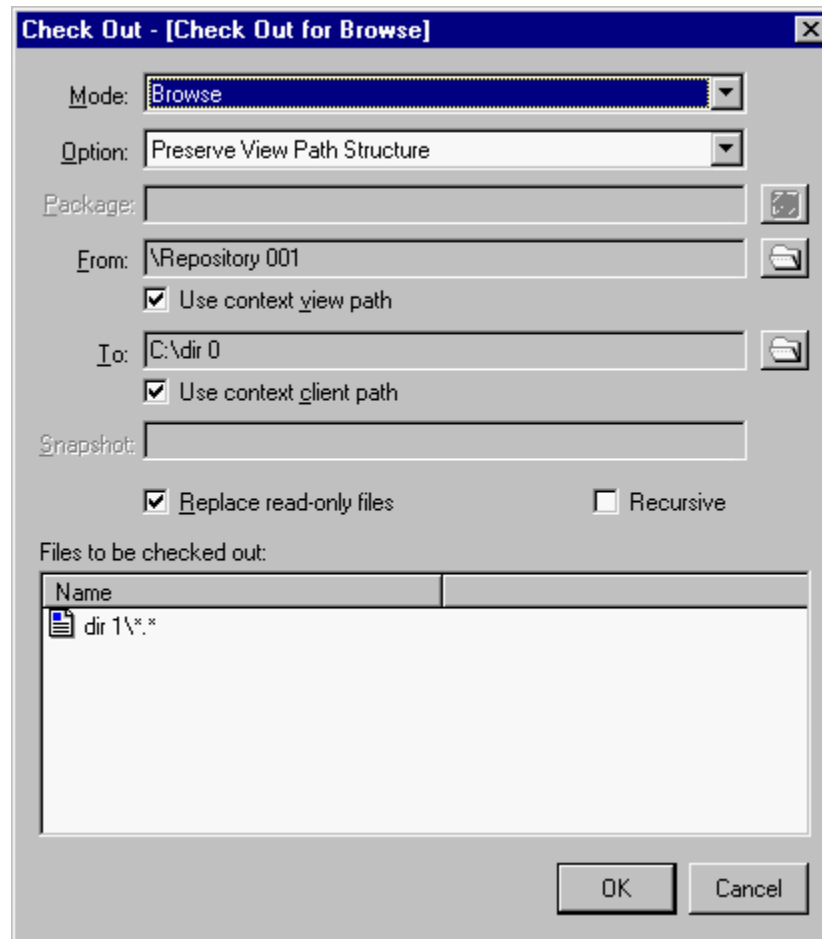
1. Go to your work area.
2. Choose Tools, Find, Files.
3. Enter the search criteria for the files you want, and then click Find Now.
4. From the search results list box, select the files you want to check in or check out.

5. Right-click one of the selected files, and then choose AllFusion Harvest CM *check in* or *check out* from the shortcut menu.



6. The check in or check out dialog displays and you can set options and execute the process.
7. You can see the results in the Message Log.

Check Out Dialog



Mode—The options on the Mode drop-down list vary, depending on how the check out process was defined during setup. One or more of the following options appears; choose **one**:

- **Update** copies selected versions to the destination client directory and creates a reserved version on each item's trunk, allowing the corresponding files to be checked back in. The permission on a read-only file is changed to normal (write access) when this mode of check out is used.
- **Browse** copies the items to the destination directory but does not allow you to check the files back in. The read-only attribute is set on files checked out for Browse mode.
- **Synchronize** determines the versions of the files in the client file system using the signature file or the VSAM dataset information, and selected versions are checked out only if they differ (either older or newer) from those in the external file system. Items are checked out in a read-only mode. The check out for Synchronize mode is especially useful in the build process.

- **Concurrent Update** copies selected versions to the destination client directory and creates a reserved version on a package branch for each item. All package updates accumulate on this branch. The permission on a read-only file is changed to normal (write access) when this mode of check out is used.
- **Reserve Only** does not move any data to external directories but creates a reserved version with a reserved tag (R) on each item's trunk so that corresponding files can be checked in.

Option—The check out options work in conjunction with the Recursive search option on the Select Version dialog. Normally, only items in one path are displayed in the dialog. When Recursive search is chosen, Harvest searches all paths beneath the current path and displays the item versions matching the other filtering criteria being used on the dialog, allowing you to select items from multiple paths for check out. Choose **one** of the following:

- **Preserve View Path** checks out all selected items into corresponding client directories, if they already exist. If the directories do not exist, an error message is displayed and the items are not checked out.
- **Preserve and Create Client Directory** checks out all selected items into matching client directories, and creates any client directories that do not currently exist.
- **All Items to Same Client Directory** places all selected items directly beneath the specified client directory, ignoring the path structure within the repository.

Package—The Package field is active when the check out mode is update, concurrent update, or reserve only. The reserved version placeholder created by this check out operation is associated with the package selected for this field. If you select a package before choosing the check out process, the Package field is automatically populated with the package name. If you do not have a package selected before invoking the check out process, you can select a package by clicking on the button next to the Package field. This displays the Select Package dialog that allows you to select packages. If only one package exists in the current state, this package is selected by default for this field.

From and **To**—These fields work together to help you synchronize the internal (repository) and external (file system) structures. They are typically used to establish a point at which paths in the repository mirror working directories on the client.

From—The view path displayed in this field specifies a location in the repository from which items are checked out. The path specified in the Set Default Context dialog determined the view path that populates this field.

Use context view path—If you select this option and you used the Set Default Context dialog to set a default view path, the view path will populate this field.

To—The directory selected for this field specifies the destination on the file system for the checked-out files. If either of the preserve structure check out options is being used, the check out specifies a path and this specification is appended to the file system directory.

If the files being checked out include a directory specification, as would be the case during recursive check out, this directory specification is appended to the client path specified here when either of the preserve structure check in options is being used.

Use context client path—If you select this option and you used the Set Default Context dialog to set a default client directory location, the file path will populate this field.

Snapshot—The Snapshot drop-down list enables you to choose a snapshot from which you can select versions to check out. This field is enabled when a snapshot exists in the view in which you are executing the check out process.

Files to be checked out – This list box shows the files/directories you selected before choosing the check out process. The content of the list box cannot be changed unless you exit the dialog, choose different files/directories on the Windows Explorer, and then choose the check out process again.

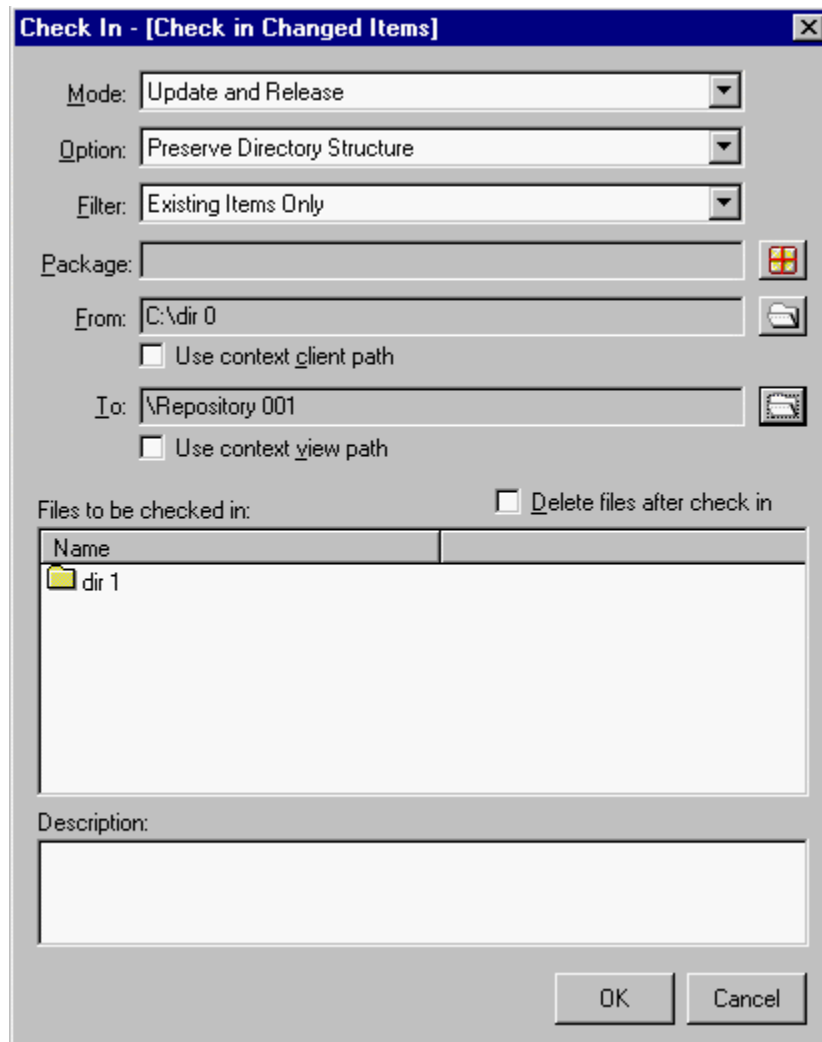
When you select an item for check out, Harvest automatically selects the item's latest versions without tags to populate the check out list box. When you select a tagged version for check out, Harvest automatically eliminates that version from your check out selection and it does not populate the check out list box. For example, if you select three versions including one that has a reserve tag and then choose check out, only the two untagged versions populate the check out list box.

Replace read-only files—This filter controls whether the checked-out files should replace existing read-only files on the file system. This allows you to replace files that you previously checked out as read-only or checked in. If no corresponding file exists on the file system, the item is checked out regardless of the value for this option.

Files that are not read-only on the PC file system, or that have write permission on the UNIX file system, are never overwritten. Harvest assumes that such a file has been checked out for update and not checked back in yet. Overwriting such a file might cause you to lose unsaved changes.

Recursive—When Recursive is selected; Harvest searches all paths beneath the current path and checks out the item versions located on the current directory and subdirectories.

Check In Dialog



Mode—A file cannot be checked in unless its corresponding item is reserved by the package performing the function. The only exception to this is items being checked in for the first time. The check in function operates in one of three modes, governed by the state of the Mode drop-down list:

- **Update and Release.** The new item or version is checked into the repository and, if the item was reserved by a package during a previous check out, the reserved tag is removed from the item. The permission on the client file is automatically set to read-only so that changes cannot be made to it until it is checked out again.
- **Update and Keep.** The item in the view is updated (or created) and the current package keeps it reserved, and the file permissions unchanged, so that more updates can be made.

- **Release Only.** No check in is performed and the item is not updated, but the item is no longer marked as reserved for the current package. The permission on the client file is automatically set to read-only so that the file cannot be changed until it is checked out again.

Check in for Release Only does not require the local file system file to exist. If the file does not exist on the local file system, you need to right-click the reserved version and then choose the check in process.

Note: Selecting Release Only mode implies an item filter of Existing Items Only because this operation requires existing reserved items in the repository; the Filters drop-down list is disabled.

Option—The Option drop-down list provides a variety of ways to check in files and filter the files being checked in. Select **one** of the following:

- **Preserve Directory Structure** checks in the selected files to repository view paths with names that correspond to their client directory location, if these view paths currently exist.
- **Preserve and Create Path Structure** checks in selected files to paths with names that correspond to their client directory location, and creates any view paths that do not currently exist.
- **All Files to Same View Path** checks in all selected files to the same path in the destination view, ignoring the client directory structure.

Filter—The Filter drop-down list provides a variety of ways to filter the files being checked in. Choose **one** of the following:

- **New or Existing Items** checks in all selected files, if they are reserved by the package or did not previously exist.
- **New Items Only** limits the check in to files that do not have corresponding items in the Harvest repository.
- **Existing Items Only** limits the check in to files that have corresponding items reserved by the package. Any files without corresponding items are skipped. This filter can be used to prevent the existence of unwanted files, such as temporary files or templates, in your repository.

Note: Only the item filters enabled in the Check In Process Properties dialog are available for selection in this field.

Package—Any updates made to an item are associated with the package selected for the Package field. If a package was specified in the Set Default Context dialog, it populates this field. If you do not have a package selected before invoking the check in process, you can select a package by clicking the button next to the Package field. This displays the Select Package dialog, which allows you to locate and select packages.

From and **To**—These fields work together to help you synchronize the internal (repository) and external (file system) structures. They are typically used to establish a point at which paths in the repository mirror working directories on the client.

From—Files are checked in from the client file directory. The client path displayed in this field is determined by your initial file selection on the Windows Explorer.

Use context client path—If you select this option and you used the Set Default Context dialog to set a default client directory location, the file path will populate this field.

To—The path selected for the To field specifies the location in the destination view for the files being checked in. If the view path was specified in the Set Default Context dialog, it populates this field. To change the path, click the button next to this field. This displays the Select Repository Item Path dialog that allows you to browse through available paths and select a location.

If the files being checked in include a directory specification, as would be the case during recursive check in, this directory specification is appended to the view path specified here when either of the preserve structure check in options is being used.

Use context view path—If you select this option and you used the Set Default Context dialog to set a default view path, the view path will populate this field.

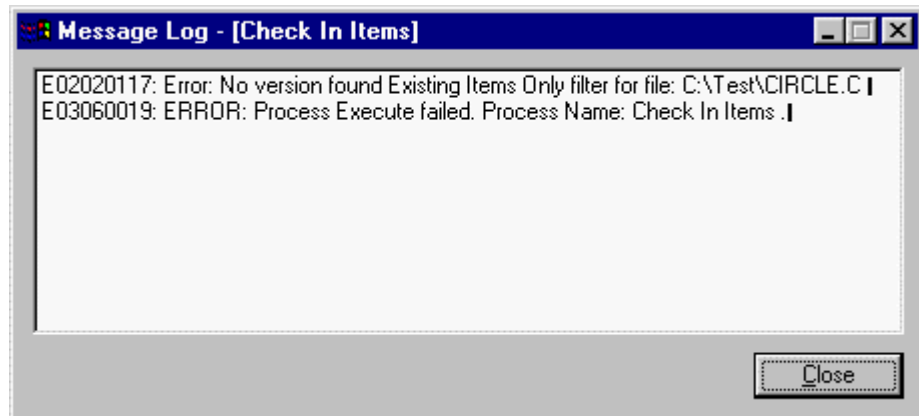
Delete files after check in—This option deletes from the client file system all files that are successfully checked in with the mode Update and release, or Release Only. Deleted files are also removed from the signature file (harvest.sig) on the client.

Files to be checked in – This list box shows the files/directories you selected before choosing the check in process. The content of the list box cannot be changed unless you exit the dialog, choose different files/directories on the Windows Explorer, and then choose the check in process again.

Description—You can type a description of the check in.

The Message Log

All Windows Extension activity is recorded in the Message Log. When you execute check in/check out, the message log window remains open for the duration of the session. The message log window is created when you log in and is closed automatically when you log out. You can use Ctrl + C to copy the content of the message log window to the clipboard.



Log Out

To log out of the Windows Extension:

1. Simply right-click any file or directory in the Windows Explorer and choose AllFusion Harvest CM, Logout.

Index

A

- Administrator window, 3-3
 - logging in, 3-2
- agent
 - options, 4-1
 - starting on Windows NT, 4-1
- application server model, 1-2
- approve process, 6-1
- architecture
 - components, 1-3
- assigning packages, 6-28

B

- base versions, 1-7
- baseline, 1-6
- baselining
 - take snapshot process, 6-53
- bind package groups
 - demote process, 6-35
- bind packages, 5-6
- branch versions, 2-5, 6-29
 - concurrent update, 2-7
 - merging, 2-8, 2-9
 - numbering, 2-7
 - rules, 2-7
- branches, 1-8, 2-5
 - multiple, 2-6
- broker, 1-3
 - changing, 3-3

C

- Change Icon dialog, 3-14
- check in process, 6-3
 - case behavior, 6-4
 - client directories, 6-6
 - client root directory, 2-2
 - default context, 3-19
 - file conversions, 6-3
 - Files tab, 4-7
 - HarWind, 9-6
 - ISPF client interface, 8-9
 - modes, 6-5
 - new items, 6-3
 - options, 6-7
 - recursive, 2-4, 3-21
 - remote, 4-7
 - view paths, 6-6
- check out process, 6-11
 - case behavior, 6-14
 - client directories, 6-16
 - concurrent update, 2-5
 - date and time stamp, 6-14
 - default context, 3-19
 - file conversions, 6-13
 - file permissions, 6-12
 - HarWind, 9-6
 - ISPF client interface, 8-12
 - modes, 6-15
 - options, 6-16
 - recursive, 2-4, 3-21
 - Replace read-only files, 6-18
 - rules, 6-11
 - synchronize, 6-11
 - versions, 6-17
 - view path root, 2-2
 - view paths, 6-16
- client directories, 6-6, 6-16

- client directory, 1-6
- client/server
 - systems, 1-1
- clients, 1-1
- command line parameters
 - User-Defined Process processes, 6-58
- communication layer, 1-3
- compare files process
 - Visual Difference dialog, 3-12
- compare views process, 6-21
 - options, 6-21
 - report options, 6-21
 - Visual Difference dialog, 6-23
- concurrent merge process, 6-25
 - options, 6-27
- concurrent update, 2-5
 - branch versions, 2-8
- context, 3-19
 - forms, 7-5
 - ISPF client interface, 8-3
 - packages, 7-5
 - setting ISPF client interface, 8-3
- create package process, 6-28
 - assigning to users, 6-28
- cross project merge process, 6-29
 - destination project, 6-30
 - options, 6-33
 - restrictions, 6-29
 - results, 6-31
 - versions, 6-31
- Customize dialog, 3-7

D

- database
 - information access, 1-1
- date and time stamp
 - check out process, 6-14
- Default Context dialog, 3-19
- delete version process, 6-34
 - limitations, 6-34
- deltas, 1-8
 - chains, 2-5

- information retained, 2-5
 - tree, 1-8
- demote process, 6-35
 - enforce package bind, 6-36
 - out of view deltas, 6-35
 - restrictions, 6-35
 - verify package dependency, 6-37
- dialogs
 - Find, 3-22
 - kinds, 3-21
 - Process execution, 3-22
 - properties, 3-22
 - Select, 3-22

E

- enforce package bind, 5-20, 6-36, 6-51
- enforce package merge, 6-51

F

- file permissions
 - check out process, 6-12
- files
 - case behavior, 6-4, 6-14
 - conversions, 6-3, 6-13
- Files tab, 3-17, 4-7
 - check in process, 4-7
 - My Computer folder, 3-17, 4-7
 - Remote Agent Neighborhood folder, 3-17, 4-7
- Find dialogs
 - executing processes, 7-1
 - File, 7-2
 - Form, 7-4
 - Package, 7-8
 - User, 7-24
 - Version, 7-12
- Find File dialog, 7-2
- Find Form dialog, 7-4
- Find Package dialog, 7-8
- Find Version dialog, 7-12
 - date, 7-16
 - executing processes, 7-12
 - recursive search, 7-15

- results, 7-18
- snapshot views, 7-12
- using multiple filters, 7-17
- form attachments, 5-13
 - adding, 5-14
 - copying, 5-16
 - removing, 5-15
 - types, 5-13
 - viewing, 5-16
- form types
 - default, 5-10
- Form Viewer, 5-9
- forms, 1-9, 5-8
 - adding attachments, 5-14
 - attachments, 5-13
 - concurrent update, 5-10
 - copying attachments, 5-16
 - default types, 5-10
 - Form Viewer, 5-9
 - move package process, 6-45
 - package associations, 5-16, 7-5
 - removing attachments, 5-15
 - viewing attachments, 5-16

H

- hagent.arg
 - options, 4-1
- Harvest
 - architecture, 1-3
 - communication layer, 1-3
 - context, 3-19
 - invoking, 3-1
 - processes, 3-5
- HarWind, 9-1
 - check in, 9-6
 - check out, 9-6
 - log out, 9-14
 - login, 9-1
 - message log, 9-14
 - set default context, 9-3
- help
 - menu, 3-11

I

- interactive merge process, 6-37
 - Auto-Merge options, 6-40
 - Conflicts menu, 6-39, 6-40
- ISPF client interface, 1-3, 8-1
 - check in process, 8-9
 - check out process, 8-12
 - context, 8-3
 - logging in, 8-1
 - packages, 8-6
 - primary panel, 8-2
 - projects, 8-4
 - settings, 8-3
 - states, 8-5
 - views, 8-7
- items, 1-7
 - new, 6-3
 - versions, 1-7

L

- life cycles
 - example, 1-11
 - package movement, 5-19
- list version process, 6-41
 - examples, 6-42
- list view, 3-18
- log out
 - HarWind, 9-14
- login
 - HarWind, 9-1

M

- mail utility
 - notify process, 6-46
- main window
 - Administrator, 3-3
 - list view, 3-18
 - menus, 3-4
 - output log, 3-18
 - toolbar, 3-12
 - workbench, 3-3
 - workspace, 3-16

- menus, 3-4
 - help, 3-11
 - Reports, 3-5
 - Tools, 3-7
- merging
 - branch versions, 2-8
 - rules, 2-9
- message log
 - HarWind, 9-14
- modes
 - check in process, 6-5
 - check out process, 6-15
- move package process, 6-44
 - package attributes, 6-45
 - package history, 6-45
 - rules, 6-44

N

- notify process, 6-46
 - mail utility, 6-46

O

- objects
 - forms, 1-9
 - hierachy, 1-4
 - package groups, 1-9
 - packages, 1-8
 - processes, 1-6
 - project, 1-5
 - repositories, 1-6
 - types, 1-4
- Options dialog, 3-10
- OS/390
 - ISPF client interface, 8-1
- output log, 3-18
 - compare views reports, 6-21
 - list version, 6-43
 - reports, 6-41

P

- package groups, 1-9, 5-6
 - bind packages, 5-6
 - properties, 5-7
- packages, 1-8
 - bind package groups, 6-49
 - bound package groups, 6-2
 - context, 7-5
 - cross project merge process, 6-29
 - enforce package bind, 5-20
 - form associations, 5-16, 7-5
 - history report, 6-45
 - ISPF client interface, 8-6
 - life cycle movement, 5-19
 - management, 5-1
 - move package process, 6-45
 - promote, 6-48
 - promote process, 6-50
 - properties, 5-2
- passwords
 - changing, 3-2
 - changing expired passwords, 3-1
 - expired, 3-1
 - policy, 3-1
 - warning, 3-1
- process, 1-6
 - process execution dialogs, 6-1
- process execution dialogs, 6-1
 - approve, 6-1
 - check in, 6-3
 - check out, 6-11
 - compare views, 6-21
 - concurrent merge, 6-25
 - create package, 6-28
 - cross project merge, 6-29
 - delete version, 6-34
 - demote, 6-35
 - interactive merge, 6-37
 - invoking, 6-1
 - list version, 6-41
 - move package, 6-44
 - notify, 6-46
 - promote, 6-48
 - remove item, 6-51
 - rename item, 6-52
 - take snapshot, 6-53
 - User-Defined Process, 6-56
- program
 - user-defined process, 6-57

Project Lifecycle Viewer, 3-17

projects

- introduction, 1-5
- ISPF client interface, 8-4

Projects tab, 3-16

- My Projects folder, 3-16
- Other Projects folder, 3-16

promote process, 6-48

- bind package groups, 6-49
- conditions, 6-49
- enforce package bind, 6-51
- enforce package merge, 6-51
- rules, 6-49
- verify package dependency, 6-51

R

recursive search, 2-4, 3-21

- Find Version dialog, 7-15

remove item process, 6-51

- considerations, 2-10
- removed tag, 2-9
- restoring an item, 6-51, 6-52

removed tags, 2-9, 6-29

rename item process, 6-52

Replace read-only files, 6-18

reports

- Administrator, 3-5
- compare views, 6-21
- List Version, 6-41
- workbench, 3-6

repositories

- introduction, 1-6
- items, 1-7
- structure, 2-2

reserved versions, 2-10

- considerations, 2-10

Resolve Form Conflicts dialog, 5-10

root

- directory, 2-2
- view path, 2-2

rules

- check out process, 6-11

S

Select a Directory Path dialog, 7-21

Select a Repository Item Path dialog, 7-23

Select a View dialog, 7-26

Select dialogs

- Directory Path, 7-21
- Package, 7-22
- Package Cross-Project, 7-22
- Repository Item path, 7-23
- View, 7-26

Select Packages Cross-Project dialog, 7-22

Select Packages dialog, 7-22

Select User dialog, 7-24

servers, 1-1

set default context

- HarWind, 9-3

settings

- ISPF client interface, 8-3

signature files, 6-4

- check in process, 6-4

snapshot views, 1-7

- Find Version dialog, 7-12

states, 1-5

- ISPF client interface, 8-5
- views, 1-5

synchronize mode, 6-11

T

tags

- merged, 2-9, 6-25
- removed, 2-9, 6-29
- reserved, 2-9

take snapshot process, 6-53

- visible to other projects, 6-55

taskbar

- changing icon, 3-14
- dialog, 3-13
- setting up, 3-13

toolbar, 3-12

toolbars, 3-8

trunks, 1-8, 2-5
 versions, 6-25

U

user groups, 1-10
 approve process, 6-1
 notify process, 6-48

User-Defined Process process, 6-56
 command line parameters, 6-58

users, 1-10
 approve process, 6-1
 notify process, 6-48

V

verify package dependency, 6-37, 6-51

versions, 1-7
 base, 1-7
 branch, 6-29
 branches, 2-5

check out process, 6-17
cross project merge process, 6-31
numbering, 2-7
removed tag, 2-9
reserved tag, 2-10
tags, 2-9
viewing, 1-8

view path, 1-6

view paths, 6-6, 6-16

views
 ISPF client interface, 8-7
 snapshot, 1-7
 working, 1-7

Visual Difference dialog, 3-12, 6-23

W

workbench, 3-3
 Files tab, 3-17, 4-7
 logging in, 3-2
 Projects tab, 3-16

working views, 1-7

workspace, 3-16